Lecture 5: Strings
(Sections 8.1, 8.2, 8.4, 8.5, 1\textsuperscript{st} paragraph of 8.9)

CS 1110
Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Fan, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]
Announcements

• No laptop use stage right (your left)
• We will use clickers, but not for credit. Therefore no need to register your clicker.
• “Partner Finding Social” Tues Feb 4th 5-6pm Gates Hall 3rd floor Lounge (1xxx-2xxx courses)
• Before next lecture, read Sections 4.9, 9.5
• To access video of lecture, log in using NetID and password “through Canvas”, but we don’t use Canvas otherwise. Course website is https://www.cs.cornell.edu/courses/cs1110/2020sp/
Today

• More about the `str` type
  ▪ New ways to use strings

• More examples of functions
  ▪ Functions with strings!

• Learn the difference between `print` and `return`
Strings are Indexed (Question 1)

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

- 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”

- \( t = 'Hello all' \)
  
  \[
  \begin{array}{c|c|c|c|c|c|c|c|c|c}
  & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
  \hline
  H & e & l & l & o & a & l & l & \text{} \\
  \end{array}
  \]

- What is \( t[3:6] \)?
  
  A: 'lo a'
  B: 'lo'
  C: 'lo '
  D: 'o '
  E: I do not know
Strings are Indexed (Solution 1)

- \( s = 'abc \text{ d}' \)
  
<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”

- \( t = 'Hello \text{ all}' \)

<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>a</td>
<td>l</td>
</tr>
</tbody>
</table>

  - What is \( t[3:6] \)?

  A: 'lo a'
  B: 'lo'
  C: 'lo'  \( \text{CORRECT} \)
  D: 'o '
  E: I do not know
Strings are Indexed (Question 2)

- \( s = 'abc d' \)
  - 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”

- \( t = 'Hello all' \)
  - What is \( t[:3] \)?
    - A: 'all'
    - B: 'l'
    - C: 'Hel'
    - D: Error!
    - E: I do not know
Strings are Indexed (Solution 2)

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

- $t = 'Hello all'$
  
  
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>b</th>
<th>l</th>
<th>o</th>
<th>a</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

  - What is $t[:3]$?
    - A: 'all'
    - B: 'l'
    - C: 'Hel'  CORRECT
    - D: Error!
    - E: I do not know
Other Things We Can Do With Strings

**Operator** \texttt{in: } \(s_1 \text{ in } s_2\)
- Tests if \(s_1\) “a part of” (or a \textit{substring} of) \(s_2\)
- Evaluates to a \texttt{bool}

**Examples:**
\begin{verbatim}
>>> s = 'abracadabra'
>>> 'a' in s
True
>>> 'cad' in s
True
>>> 'foo' in s
False
\end{verbatim}

**Built-in Function** \texttt{len: } \texttt{len(s)}
- Value is \# of chars in \(s\)
- Evaluates to an \texttt{int}

**Examples:**
\begin{verbatim}
>>> s = 'abracadabra'
>>> len(s)
11
>>> len(s[1:5])
4
>>> s[1:len(s)-1]
'bracadabr'
\end{verbatim}
Defining a String Function

Want to write function `middle`, which returns the middle 3\textsuperscript{rd} of a string (length divisible by 3).

How we want it to behave:

```python
>>> middle('abc')
'b'
>>> middle('aabbcc')
'bb'
>>> middle('aaabbbccc')
'bbb'
```

Important Questions:

1. What are the parameters?
2. What is the return value?
3. What goes in the body?

```python
def middle(text):
    ???
    return middle_third
```
Steps to writing a program

1. Work an instance yourself
2. Write down exactly what you just did
3. Generalize your steps from 2
4. Test your steps
5. Translate to Code
6. Test program
7. Debug (if necessary)
Steps to writing a program

1. Work an instance yourself
2. Write down exactly what you just did
3. Generalize your steps from 2
4. Test your steps
5. Translate to Code

```python
>>> middle('abc')  # middle_third = text[1]  # Too easy!!
>>> middle('aabbee')  # middle_third = text[2:4]  # Still too easy!!
```

```python
>>> middle('It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way…')
```
Definition of middle

def middle(text):
    """Returns: middle 3\textsuperscript{rd} of text
    Param text: a string with
    length divisible by 3""

IMPORTANT:
Precondition requires
that arguments to
middle have length
divisible by 3.

If not? Bad things
could happen, and we
blame the user (not
the author) of the
function.
def middle(text):
    """Returns: middle 3rd of text
Param text: a string with length divisible by 3"

    # Get length of text
    size = len(text)
    # Start of middle third
    start2 = size//3
    # End of middle third
    start3 = (2*size)//3
    # Get the substring
    middle_third = text[start2:start3]
    return middle_third

IMPORTANT:
Precondition requires that arguments to `middle` have length divisible by 3.
If not? Bad things could happen, and we blame the user (not the author) of the function.
Advanced String Features: Method Calls

- Strings have some useful *methods*
  - Like functions, but “with a string in front”
- **Format:** `<string name> . <method name> (x, y, . . .)`
- **Example:** `upper()` returns an upper case version
  ```python
  >>> s = 'Hello World'
  >>> s.upper()
  'HELLO WORLD'
  >>> s[1:5].upper()
  'ELLO'
  >>> 'scream'.upper()
  'SCREAM'
  >>> 'cs1110'.upper()
  'CS1110'
  ```
Examples of String Methods

- **s1.index(s2)**
  - Returns position of the first instance of \( s_2 \) in \( s_1 \)
  - **error** if \( s_2 \) is not in \( s_1 \)

- **s1.count(s2)**
  - Returns number of times \( s_2 \) appears inside of \( s_1 \)

- **s.strip()**
  - Returns a copy of \( s \) with white-space removed at ends

- \( s = 'abracadabra' \)

<table>
<thead>
<tr>
<th></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>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>r</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>r</td>
<td>a</td>
</tr>
</tbody>
</table>

  - **s.index('a')**
    - 0
  - **s.index('rac')**
    - 2
  - **s.count('a')**
    - 5
  - **s.count('b')**
    - 2
  - **s.count('x')**
    - 0
  - **'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 ()""

>>> s = 'One (Two) Three'
>>> firstparens(s)
'Two'

>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()"""

    # Find the open parenthesis
    start = text.index('(')

    # Find the close parenthesis
    end = text.index(')')

    inside = text[start+1:end]

    return inside

>>> s = 'One (Two) Three'
>>> firstparens(s)
'Two'

>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
Steps to writing a program

1. Work an instance yourself
2. Write down exactly what you just did
3. Generalize your steps from 2
4. Test your steps
5. Translate to Code
6. **Test program**  *Think of all the corner cases*
7. Debug (if necessary)  *What could possibly go wrong?*
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()""
    # Find the open parenthesis
    start = text.index('(')
    # Store part AFTER paren
    substr = text[start+1:]
    # Find the close parenthesis
    end = substr.index(')')
    inside = substr[:end]
    return inside

>>> s = 'One (Two) Three'
>>> firstparens(s)
'Two'

>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
def second(thelist):
    """Returns: second word in a list
    of words separated by commas, with
    any leading or trailing spaces from the
    second word removed
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words with
    at least two commas """

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

Is there an error?

A: Yes, Line 1
B: Yes, Line 2
C: Yes, Line 3
D: Yes, Line 4
E: There is no error
def second(thelist):
    """Returns: second word in a list of words separated by commas, with any leading or trailing spaces from the second word removed
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words with at least two commas """
    start = thelist.index((',',)
    tail = thelist[start+1:]
    end = tail.index((',',)
    result = tail[:end]
    return result

>>> second('cat, dog, mouse, lion')
expecting: 'dog' get: ' dog'

>>> second('apple, pear, banana')
expecting: 'pear' get: ' pear'

Is there an error?

A: Yes, Line 1
B: Yes, Line 2
C: Yes, Line 3
D: Yes, Line 4
E: There is no error
def second(thelist):
    """Returns: second word in a list of words separated by commas, with any leading or trailing spaces from the second word removed
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words with at least two commas """

    # Split the list at the first comma
    start = thelist.index(',')
    tail = thelist[start+1:]
    end = tail.index((',', ')
    result = tail[:end]
    return result

>>> second('cat, dog, mouse, lion')
expecting: 'dog' get: 'dog'

>>> second('apple,pear , banana')
expecting: 'pear' get: 'pear'

def second(thelist):
    """Returns: second word in a list
    of words separated by commas, with
    any leading or trailing spaces from the
    second word removed
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words with
    at least two commas """
    start = thelist.index((',',)
    tail = thelist[start+1:]       #possible fix ??
    end = tail.index(',',)
    result = tail[:end].strip()   #better fix!
    return result

>>> second('cat, dog, mouse, lion')
expecting: 'dog'       get: ' dog'

>>> second('apple,pear , banana')
expecting: 'pear'       get: 'pear '
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: \ followed by letter (character)
  - 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>