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

CS 1110
Introduction to Computing Using Python

http://www.cs.cornell.edu/courses/cs1110/2018sp

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

Having trouble finding Piazza?

• New link on our webpage should help!
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)

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

- 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”

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

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

- \( s = 'abc\ d' \)
  - 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 \( t[3:6] \)?
    - A: 'lo a'
    - B: 'lo'
    - C: 'lo' CORRECT
    - D: 'o'
    - E: I do not know
Strings are Indexed (Question 2)

- \( s = 'abc d' \)
  
<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”

- \( t = 'Hello all' \)
  
<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>a</td>
<td>l</td>
<td>l</td>
</tr>
</tbody>
</table>

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

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

- 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”

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

- 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 in: s₁ in s₂**

- Tests if s₁ “a part of” (or a *substring* of) s₂
- Evaluates to a bool

**Examples:**

```python
>>> s = 'abracadabra'
>>> 'a' in s
True
>>> 'cad' in s
True
>>> 'foo' in s
False
```

**Built-in Function len: len(s)**

- Value is # of chars in s
- Evaluates to an int

**Examples:**

```python
>>> s = 'abracadabra'
>>> len(s)
11
>>> len(s[1:5])
4
>>> s[1:len(s)-1]
'bracadabr'
>>> 
```
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

>>> middle('abc')
middle_third = text[1]  Too easy!!

>>> middle('aabbee')
middle_third = text[2:4]  Still too easy!!

>>> 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...')
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

- \texttt{s} \texttt{1.index(s} \texttt{2)}
  - Returns position of the first instance of \texttt{s} \texttt{2} in \texttt{s} \texttt{1}
  - \texttt{error} if \texttt{s} \texttt{2} is not in \texttt{s} \texttt{1}

- \texttt{s} \texttt{1.count(s} \texttt{2)}
  - Returns number of times \texttt{s} \texttt{2} appears inside of \texttt{s} \texttt{1}

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

- \texttt{s = 'abracadabra'}

- \texttt{s.index('a')} 0
- \texttt{s.index('rac')} 2
- \texttt{s.count('a')} 5
- \texttt{s.count('b')} 2
- \texttt{s.count('x')} 0
- \texttt{'}ab\texttt{'.strip()} \texttt{'a\texttt{ 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 = text.index('(')

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

    return text[start+1:end]
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(')')
    return substr[:end]

>>> 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

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

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

Where is the error?

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

```python
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 ' 
```

What if there are multiple (or no!) spaces?
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; &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`
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 (returns None)
<table>
<thead>
<tr>
<th>print</th>
<th>vs.</th>
<th>return</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Displays a value on screen</td>
<td>• Sends a value from a function call frame back to the caller</td>
<td></td>
</tr>
<tr>
<td>• Used primarily for <strong>testing</strong></td>
<td>• Important for <strong>calculations</strong></td>
<td></td>
</tr>
<tr>
<td>• Not useful for calculations</td>
<td>• Does not display anything</td>
<td></td>
</tr>
</tbody>
</table>

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

>>> print_plus(2)
3

>>> return_plus(2)
3
```
Python Interactive Mode

• executes both *statements* and *expressions*

• *if expression:*
  1. *evaluates*
  2. *prints value (if one exists)*

```python
>>> 2+2
4
```
evaluates (performs addition)

```python
>>> return_plus(2)
3
```
evaluates (makes function call, gets return value)
**return_plus** in action

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

1. Evaluates: makes function call, evaluates to return value
2. prints value
Python Interactive Mode

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

```
>>> print_plus(2)
3
>>> 
```

**print_plus in action**

1. Evaluates: makes function call, evaluates to return value
2. does not print value b/c it’s NONE
hybrid_plus in action

def hybrid_plus(n):
    print(n)
    return n+1

Python Interactive Mode

>>> print_plus(2)
2
3

1. Evaluates: makes function call, evaluates to return value
2. print value
See the difference in the Python Tutor

```python
def print_plus(n):
    print(n+1)
def return_plus(n):
    return n+1
x1 = print_plus(2)
x2 = return_plus(2)
print(x1)
print(x2)
```

Program output:

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>None</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

http://cs1110.cs.cornell.edu/visualizer/#mode=edit
Module Text

# module.py

def foo(x):
    x = 1+2
    x = 3*x

Python Interactive Mode

>>> import module
>>> print(module.x)

... What does Python give me?

A: 9
B: 10
C: 1
D: None
E: Error
# Exercise 1, Solution

## Module Text

```python
# module.py

def foo(x):
    x = 1 + 2
    x = 3 * x
```

## Python Interactive Mode

```python
>>> import module
>>> print(module.x)
... What does Python give me?
```

A: 9
B: 10
C: 1
D: None
E: Error  **CORRECT**
Exercise 2

Module Text

```python
# module.py

def foo(x):
    x = 1+2
    x = 3*x

y = foo(0)
```

Python Interactive Mode

```bash
>>> import module
>>> print(module.y)
```

What does Python give me?

A: 9  
B: 10  
C: 1  
D: None  
E: Error
### Exercise 2, Solution

<table>
<thead>
<tr>
<th>Module Text</th>
<th>Python Interactive Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td># module.py</td>
<td>&gt;&gt;&gt; import module</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt;&gt; print(module.x)</td>
</tr>
<tr>
<td>def foo(x):</td>
<td>...</td>
</tr>
<tr>
<td>x = 1+2</td>
<td>What does Python give me?</td>
</tr>
<tr>
<td>x = 3*x</td>
<td></td>
</tr>
<tr>
<td>x = foo(0)</td>
<td></td>
</tr>
</tbody>
</table>

A: 9
B: 10
C: 1
D: None  **CORRECT**
E: Error
Exercise 3

Module Text

```python
# module.py

def foo(x):
    x = 1+2
    x = 3*x
    return x+1

y = foo(0)
```

Python Interactive Mode

```python
>>> import module
>>> module.y
```

```
What does Python give me?
```

A: 9
B: 10
C: 1
D: None
E: Error
**Exercise 3, Solution**

<table>
<thead>
<tr>
<th>Module Text</th>
<th>Python Interactive Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td># module.py</td>
<td></td>
</tr>
</tbody>
</table>
| def foo(x):
  x = 1+2
  x = 3*x
  return x+1 |
| y = foo(0) |
| >>> import module |
| >>> module.y |
| ... | What does Python give me? |
| A: 9 |
| B: 10 CORRECT |
| C: 1 |
| D: None |
| E: Error |
Exercise 4

Function Definition

def foo(a,b):
  x = a
  y = b
  return x*y+y

Function Call

>>> x = 2
>>> foo(3,4)
>>> x
...

What does Python give me?

A: 2
B: 3
C: 16
D: None
E: I do not know
## Exercise 4, Solution

### Function Definition

```python
def foo(a, b):
    x = a
    y = b
    return x * y + y
```

### Function Call

```python
>>> x = 2
>>> foo(3, 4)
>>> x
```

What does Python give me?

<table>
<thead>
<tr>
<th>A</th>
<th>2</th>
<th>CORRECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>I do not know</td>
<td></td>
</tr>
</tbody>
</table>