Lecture 5

Strings

[Andersen, Gries, Lee, Marschner, Van Loan, White]
Today

• Return to the string (str) type
  ▪ Learn several new ways to use strings
• See more examples of functions
  ▪ Particularly functions with strings
• Learn the difference between print and return
Strings are Indexed

- s = 'abc d'

  0 1 2 3 4
  a b c 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'

  0 1 2 3 4 5 6 7 8
  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

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

- 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”

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

- 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

• **Operation** `in: s₁ in s₂`
  - Tests if `s₁` “a part of” `s₂`
  - Say `s₁` a *substring* of `s₂`
  - 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

```python
>>> middle('abc')
'b'

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

>>> middle('aaabbbcccc')
'bbb'
```
Defining a String Function

1. Add string parameter
2. Add return at end
   - Set to be “result” for now
3. Work in reverse
   - Set subgoals
   - Identify needed operations
   - Store results in variables
   - Assign on previous lines

```python
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
    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

```python
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
    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'
```
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()` - converts to upper case
  ▪ `s = 'Hello World'`
  ▪ `s.upper()``HELLO WORLD`
  ▪ `s[1:5].upper()``ELLO`
  ▪ `'methods'.upper()``METHODS`
  ▪ `'cs1110'.upper()``CS1110`
Examples of String Methods

- \texttt{s1.index(s2)}
  - Position of the first instance of \texttt{s2} in \texttt{s1}

- \texttt{s1.count(s2)}
  - Number of times \texttt{s2} appears inside of \texttt{s1}

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

- \texttt{s = 'abra\text{cada\text{gram'}}}
- \texttt{s.index('a')} \hspace{1cm} 0
- \texttt{s.index('rac')} \hspace{1cm} 2
- \texttt{s.count('a')} \hspace{1cm} 5
- \texttt{s.count('b')} \hspace{1cm} 2
- \texttt{s.count('x')} \hspace{1cm} 0
- \texttt{' a b '.strip()} \hspace{1cm} '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 = text.index('(')
    # Store part AFTER paren
    substr = text[start+1:]
    # Find the close parenthesis
    end = substr.index(')')
    # Return the result
    return substr[:end]

>>> 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 = s.index('(')
    # Store part AFTER paren
    tail = s[start+1:]
    # Find the close parenthesis
    end = tail.index(')')
    # Return the result
    return tail[: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 and spaces.
    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')
'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
def second(thelist):
    ""
    Returns: second word in a list of words separated by commas and spaces.
    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')
'dog'
>>> second('apple, pear, banana')
'pear'

but what if there are multiple spaces?
result = tail[:end].strip()
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
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

    Displays these strings on the screen
Procedures vs. Fruitful Functions

**Procedures**
- Functions that **do** something
- Call them as a **statement**
- Example: 
  
  greet('Prof. Andersen')

**Fruitful Functions**
- Functions that give a **value**
- Call them in an **expression**
- Example:
  
  x = round(2.56,1)
print vs. return

- Sometimes appear to have similar behavior

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

>>> print_plus(2)
3

```
<table>
<thead>
<tr>
<th>Print</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Displays a value on the screen</td>
<td>• Sends a value from a function call frame back to the caller</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 Interactive Shell

- executes both statements and expressions
- if expression, prints value (if it exists)

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

```python
>>> 2+2
4
prints to screen

>>> return_plus(2)
3
prints to screen
```
So why do these behave similarly?

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

>>> return_plus(2)
3

def print_plus(n):
    print n+1

>>> print_plus(2)
3
```

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

>>> return_plus(2)
3
```
def return_plus(n):
    return n+1

>>> return_plus(2)
3

Shell automatically prints expression value

call frame

expression

returns value

creates
def print_plus(n):
    print n+1

print_plus(2)
3

Python Interactive Shell

>>> print_plus(2)
3

Shell tries to print expression value but there is no value (because no return!)
## print vs. return

<table>
<thead>
<tr>
<th>Print</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>def print_plus(n):</strong></td>
<td><strong>def return_plus(n):</strong></td>
</tr>
<tr>
<td>print n+1</td>
<td>return n+1</td>
</tr>
<tr>
<td>&gt;&gt;&gt;&gt; x = print_plus(2)</td>
<td>&gt;&gt;&gt;&gt; x = return_plus(2)</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Nothing here!