Lecture 5

Strings
# Announcements For This Lecture

## Assignment 1
- Will post it on Sunday
  - Need one more lecture
  - But start *reading* it
- Due Wed Sep. 25th
  - Revise until correct
  - Final version Oct 2nd
- Do not put off until end!

## Getting Help
- Can work in pairs
  - Will set up
  - Submit one for both
- Lots of consultant hours
  - Come early! Beat the rush
  - Also use TA office hours
- One-on-Ones next week
One-on-One Sessions

- Starting **Monday**: 1/2-hour one-on-one sessions
  - Bring computer to work with instructor, TA or consultant
  - Hands on, dedicated help with Lab 3 (or next lecture)
  - To prepare for assignment, **not for help on assignment**
- **Limited availability**: we cannot get to everyone
  - Students with experience or confidence should hold back
- **Sign up online in CMS**: first come, first served
  - Choose assignment One-on-One
  - Pick a time that works for you; will add slots as possible
  - Can sign up starting at 5pm **TOMORROW**
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**: "Don't" or '6" tall'
  - 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>

```python
>>> x = 'I said: "Don\'t"'
>>> print(x)
I said: "Don't"
```
String 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”

- s = 'Hello all'
  
  0 1 2 3 4 5 6 7 8
  H e l l o a l l

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

- \( s = 'abc\ d' \)

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

- 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”

- \( s = 'Hello\ all' \)

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
H & e & l & l & o & \ & \ & \ & all \\
\end{array}
\]

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

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

  - 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”

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

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

- **Operation** `in`: `s_1 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

```python
def middle(text):
    # Get length of text
    # Start of middle third
    # End of middle third
    # Get the text
    # Return the result
    return result
```

9/12/19  Strings  11
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

---

def middle(text):
    
    """Returns: middle 3\text{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
```

9/12/19 Strings
Defining a String Function

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

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

>>> middle('aaabbbccc')
'bbb'

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

**Historical Aside**

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

### Print

- Displays a value on screen
  - Used primarily for **testing**
  - Not useful for calculations

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

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

### Return

- Defines a function’s value
  - Important for **calculations**
  - But does not display anything

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

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

---

9/12/19 Strings 19
# Print vs. Return

## Print

- Displays a value on screen
  - Used primarily for **testing**
  - Not useful for calculations

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

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

## Return

- Defines a function’s value
  - Important for **calculations**
  - But does not display anything

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

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

---

```
>>> x
3
```

---

**Nothing here!**
Method Calls

- Methods calls are unique (right now) to strings
  - Like a function call with a “string in front”
- **Method calls** have the form
  \[
  \text{string.name}(x,y,\ldots)
  \]
  - The string in front is an **additional** argument
    - Just one that is not inside of the parentheses
    - **Why?** Will answer this later in course.
Example: **upper()**

- **upper()**: Return an upper case **copy**
  
  ```python
  >>> s = 'Hello World'
  >>> s.upper()
  'HELLO WORLD'
  >>> s[1:5].upper()    # Str before need not be a variable
  'ELLO'
  >>> 'abc'.upper()      # Str before could be a literal
  'ABC'
  ```

- Notice that *only* argument is string in front
Examples of String Methods

- **s₁.index(s₂)**
  - Returns position of the *first* instance of s₂ in s₁

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

- **s.strip()**
  - Returns copy of s with no white-space at *ends*

```python
>>> s = 'abracadabra'
>>> s.index('a')
0
>>> s.index('rac')
2
>>> s.count('a')
5
>>> s.count('x')
0
>>> '  a b '.strip()
'a b'
```
Examples of String Methods

- **s1.index(s2)**
  - Returns position of the first instance of s2 in s1

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

- **s.strip()**
  - Returns copy of s with no white-space at ends

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

See Lecture page for more
Working on Assignment 1

• You will be writing a lot of string functions
• You have three main tools at your disposal
  ▪ **Searching**: The index method
  ▪ **Cutting**: The slice operation `[start:end]`
  ▪ **Gluing**: The `+` operator
• Can combine these in different ways
  ▪ Cutting to pull out parts of a string
  ▪ Gluing to put back together in new string
def firstparens(text):
    
    """Returns: substring in ()
Uses the first set of parens
Param text: a string with ()"

    # SEARCH for open parens
    start = text.index('(')
    # CUT before paren
    tail = text[start+1:]
    # SEARCH for close parens
    end = tail.index(')')
    # CUT and return the result
    return tail[:end]

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

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

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

>>> second('apple, pear, banana')
'pear'
def second(text):
    """Returns: second elt in text
    The text is a sequence of words separated by commas, spaces.
    Ex: second('A, B, C') rets 'B'
    Param text: a list of words""
    start = text.index(',')  # SEARCH
    tail = text[start+1:]    # CUT
    end = tail.index(',') # SEARCH
    result = tail[:end] # CUT
    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(text):
    """Returns: second elt in text
    The text is a sequence of words
    separated by commas, spaces.
    Ex: second('A, B, C') rets 'B'
    Param text: a list of words"
    start = text.index(',')  # SEARCH
    tail = text[start+1:]    # CUT
    end = tail.index(',') # SEARCH
    result = tail[:end] # CUT
    return result

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

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

1  start = text.index(',',)  # SEARCH
2  tail = text[start+1:]  # CUT
3  end = tail.index(',',)  # SEARCH
4  result = tail[:end]  # CUT
5  return result

OR

4  result = tail[:end].strip()