Lecture 4

Strings

Announcements For This Lecture

Readings

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

Next Lab

- Strings
- **Testing functions**

Assignment 1

- Will post it on Monday
  - Need one more lecture
- Due Thu, Feb. 23nd
  - Lab 4 gives time to work
  - Revise until correct
- Can work in pairs
  - Submit one for both

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

Type: str

Char	Meaning
\'	single quote
\"	double quote
\n	new line
\t	tab
\\	backslash

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'

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 '   **CORRECT**
  D: 'o '
  E: I do not know

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[:4]?

  A: 'o all'
  B: 'Hello'
  C: 'Hell'
  D: Error!
  E: I do not know

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[:4]?

- s[:-1]

  > 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):
    """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
```

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

    # 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 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

    # 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
```

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
```

Not All Functions Need a Return

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?'

Displays these strings on the screen

No assignments or return
The call frame is **EMPTY**

Procedures vs. Fruitful Functions

Procedures

- Functions that **do** something
- Call them as a **statement**
- Example: greet('Walker')

Functions

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

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

```
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)
>>>
```

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
>>>
```

x

Nothing here!

Return

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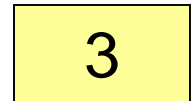 - Important for **calculations**
  - But does not display anything

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

x | 3

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

String Extraction Example

```python
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'
```

String Extraction Puzzle

```python
def second(thelist):
    """Returns: second item 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"""
```

1	`start = thelist.index(',')`	`>>> second('cat, dog, mouse, lion')`
2	`tail = thelist[start+1:]`	`'dog'`
3	`end = tail.index(',')`	`>>> second('apple, pear, banana')`
4	`result = tail[:end]`	`'pear'`
5	`return result`	

String Extraction Puzzle

```
def second(thelist):
    """Returns: second item 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"""
1   start = thelist.index(',')
2   tail = thelist[start+1:]
3   end = tail.index(',')
4   result = tail[:end]
5   return result
```

Where is the error?

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

```
>>> second('cat, dog, mouse, lion')
'dog'
>>> second('apple, pear, banana')
'pear'
```

String Extraction Puzzle

```
def second(thelist):
    """Returns: second item 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"""
```

tail = thelist[start+2:]

OR

result = tail[:end].strip()

```
1   start = thelist.index(',')
2   tail = thelist[start+1:]
3   end = tail.index(',')
4   result = tail[:end]
5   return result
```

>>> second('cat, dog, mouse, lion')
'dog'
>>> second('apple, pear, banana')
'pear'