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
  - "abo d" (Python prefers)
  - "abo 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

String are Indexed

- s = 'abo d'
  - 0 1 2 3 4
  - Access characters with []
    - s[0] is 'a'
    - s[4] is 'd'
    - s[5] causes an error
    - s[2] is 'b' (excludes c)
    - s[3] is 'o d'
  - Called “string slicing”
- s = 'Hello all'
  - 0 1 2 3 4 5 6 7 8
  - What is s[3:6]?

<table>
<thead>
<tr>
<th>Char</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>single quote</td>
</tr>
<tr>
<td>'</td>
<td>double quote</td>
</tr>
<tr>
<td>\n</td>
<td>new line</td>
</tr>
<tr>
<td>\</td>
<td>tab</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
</tbody>
</table>

Other Things We Can Do With Strings

- Operation in: s1 in s2
  - Tests if s1 is a part of s2
  - Say s1 is 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[1:8]) == 4
    - s[1:len(s)-1] == 'bracadabra'

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
```

Not All Functions Need a Return

```python
def greet(n):
    """Print 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

<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: <code>greet('Walker')</code></td>
<td>Example: <code>x = round(2.56,1)</code></td>
</tr>
</tbody>
</table>

### Historical Aside
- Historically “function” = “fruitful function”
- But now we use “function” to refer to both

## Print vs. Return

<table>
<thead>
<tr>
<th>Print</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>Displays a value on screen</td>
<td>Defines a function’s value</td>
</tr>
<tr>
<td>Used primarily for testing</td>
<td>Important for calculations</td>
</tr>
<tr>
<td>Not useful for calculations</td>
<td>But does not display anything</td>
</tr>
</tbody>
</table>

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

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

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

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

## 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[1:8].upper()` == 'ELLO'
  - `'abc'.upper()` == 'ABC'

Will see why we do it this way later in course

## Examples of String Methods

- `s1.index(s2)`
  - Position of the first instance of `s2` in `s1`
- `s1.count(s2)`
  - Number of times `s2` appears inside of `s1`
- `s.strip()`
  - A copy of `s` with white-space removed at ends

- `s = 'abracadabra'
  - `s.index('a')` = 0
  - `s.index('r')` = 2
  - `s.count('a')` = 5
  - `s.count('b')` = 2
  - `s.count('x')` = 2
  - `'abc'.strip()` == 'abc'

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]
```

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

## String Extraction Puzzle

```python
def second(thelist):
    # Returns: second elt 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

    # Find the start
    start = thelist.index(',')
    # Find the end
    end = start+1
    # Return the result
    return thelist[end]
```

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

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