Announcements

Reminders:
- Program 6 – due tomorrow by 11:55pm
- Midterm 2 on Nov. 8th

String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>endswith(substring)</code></td>
<td>The substring argument is a string. The method returns true if the string ends with substring.</td>
</tr>
<tr>
<td><code>find(substring)</code></td>
<td>The substring argument is a string. The method returns the lowest index in the string where substring is found. If substring is not found, the method returns -1.</td>
</tr>
<tr>
<td><code>replace(old, new)</code></td>
<td>The old and new arguments are both strings. The method returns a copy of the string with all instances of old replaced by new.</td>
</tr>
<tr>
<td><code>startswith(substring)</code></td>
<td>The substring argument is a string. The method returns true if the string starts with substring.</td>
</tr>
</tbody>
</table>

Using the `find` method

```python
def main():
    filename = "First_Last_assignment_file_lastname_firstname_prg6.py"
    print(renamefile(filename))

def renamefile(filename):
    ind = filename.find("file_")
    filename = filename[ind+len("file_")]
    return filename

main()
```

Output:

`lastname_firstname_prg6.py`
Testing, Searching, and Manipulating Strings

- You can use the `in` operator to determine whether one string is contained in another string
  - General format: `string1 in string2`
    - `string1` and `string2` can be string literals or variables referencing strings
- Similarly, you can use the `not in` operator to determine whether one string is not contained in another string

Class Practice

- Write a function called `count_unique` that counts the number of unique characters in a string.
  - `count_unique("abracadabra")` returns 5.
- Write a function called `count_dups` that counts the number of back-to-back duplicated characters in a string.
  - `count_dups("balloon")` returns 2

Introduction to Lists

- **List**: an object that contains multiple data items
  - **Element**: An item in a list
  - **Format**: `list = [item1, item2, etc.]`
  - Can hold items of different types
- `print` function can be used to display an entire list
- `list()` function can convert certain types of objects to lists

Introduction to Lists

A list of integers
  - `even_numbers = [2, 4, 6, 8, 10]`

A list of strings
  - `names = ['Molly', 'Steven', 'Will', 'Alicia']`

A list holding different types
  - `info = ['Alicia', 27, 1550.87]`
Example Using Lists

```python
def main():
    # Create a list with some items.
    food = ['Pizza', 'Burgers', 'Chips']
    # Display the list.
    print('Here are the items in the food list:')
    print(food)

    # Call the main function.
    main()
```

Why use lists?

- Lists exist so programmers can store multiple related variables together.
- Useful when we don't know ahead of time how many items we are going to store.
  - Lists solve this problem because a single list can hold from zero to practically any number of items in it.

Basic list operations

- Lists are created using square brackets around items separated by commas.
  ```python
  mylist = [1, 2, 3]
  numbers = [-9.1, 4.77, 3.14]
  fred = ['happy', 'fun', 'joy']
  ```
- Lists are accessed using indices/positions just like strings.
- Most (but not all) string functions also exist for lists.

<table>
<thead>
<tr>
<th>Strings</th>
<th>Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>string_var = &quot;abc123&quot;</code></td>
<td><code>list_var = [item1, item2, ...]</code></td>
</tr>
<tr>
<td><code>string_var = &quot;&quot;</code></td>
<td><code>list_var = []</code></td>
</tr>
<tr>
<td><code>len(&quot;abc123&quot;)</code></td>
<td><code>len([3, 5, 7, 9])</code></td>
</tr>
<tr>
<td><code>len(string_var)</code></td>
<td><code>len(list_var)</code></td>
</tr>
<tr>
<td><code>string_var[p]</code></td>
<td><code>list_var[p]</code></td>
</tr>
<tr>
<td><code>string_var[p:q]</code></td>
<td><code>list_var[p:q]</code></td>
</tr>
<tr>
<td><code>str3 = str1 + str2</code></td>
<td><code>list3 = list1 + list2</code></td>
</tr>
<tr>
<td><code>str3 = &quot;abc&quot; + &quot;def&quot;</code></td>
<td><code>list3 = [1, 2, 3] + [4, 5, 6]</code></td>
</tr>
<tr>
<td>&quot;i&quot; in &quot;team&quot; -&gt; False</td>
<td>7 in [2, 4, 6, 8] -&gt; False</td>
</tr>
</tbody>
</table>
One important difference

Strings are **immutable**
- You can’t change a string without making a copy of it.
  
  ```
  s = "abc"
  s[0] = "A"    # illegal!
  s = "A" + s[1:]    # legal
  ```

Lists are **mutable**
- Can be changed “in-place” (without explicit copying)
  
  ```
  L = [2, 4, 6, 8, 10]
  L[0] = 15    # legal
  L.append(26)    # legal
  ```

Compare Immutable and Mutable

- How can we switch the first and last letter in a string?
- How can we switch the first and last items in a list?

Three common ways to make a list

- Make a list that already has stuff in it:
  ```
  lst = [4, 7, 3, 8]
  ```

- Make a list of a certain length that has the same element in all positions:
  ```
  lst = [0] * 4    # makes the list [0,0,0,0]
  ```
  - Common when you need a list of a certain length ahead of time.
  - Uses the repetition operator, similarly to strings

- Make an empty list:
  ```
  lst = []
  ```
  - Common when you’re going to put things in the list coming from the user or a file.