Using `len` function

```python
# Prints 1 letter of city on each line
city = 'Boston'
index = 0
while index < len(city):
    print(city[index])
    index += 1
```

# Equivalent Code
```
city = 'Boston'
for index in range(0, len(city)):
    print(city[index])
```

Accessing Characters Review

Strings are stored character by character. Each character in a string is numbered by its position:

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>'C'</td>
<td>'o'</td>
<td>'m'</td>
<td>'p'</td>
<td>'u'</td>
<td>'t'</td>
<td>'e'</td>
<td>'r'</td>
</tr>
</tbody>
</table>
```

The numbers shown here above the characters are called *indices* (singular: index) or *positions*.
Negative Indices

Negative indexing can be used. Particularly useful for getting characters near the end of a string.

```
-8 -7 -6 -5 -4 -3 -2 -1
  "C" "o" "m" "p" "u" "t" "e" "r"
```

s[2] is the same as s[-6] both refer to “m”

To find last letter in string use:

s[-1]

String Indices

- Two ways to use square brackets
  - 1 number inside -> gives you 1 character of a string
    - s[0] gives you the first character in s
    - If s = “Computer”, s[0] gives you ‘C’
  - 2 numbers inside (separated by a colon) -> gives you a substring or string slice

String Slicing

- **Slice**: span of items taken from a sequence, known as substring
  - Slicing format: `string[start : end]`
    - Expression will return a string containing a copy of the characters from `start` up to, but not including, `end`
    - If `start` not specified, 0 is used for start index
    - If `end` not specified, `len(string)` is used for end index
  - Slicing expressions can include a step value and negative indexes relative to end of string
  
```
s[0:1] -> “C” just like s[0]
s[0:2] -> “Co”
s[0:7] -> “Computer”
s[3:6] -> “put”
s[0:8] -> “Computer”
```
Indices Don’t have to be Literal Numbers

Say we have this code:
s = input("Type in a string: ")
x = int(len(s) / 2)
print s[0:x])

What does this print?

More Fun with Indices

• Examples using negative indices
• A negative index counts from the right side of the string, rather
  than from the left
s = “Computer”
print(s[-1])  # prints r
print(s[3:len(s)])  # prints ter
print(s[1:-1])  # prints ompute

More Fun with Indices

• Slices don’t need both left and right indices
• Missing left -> use 0 [far left of string]
• Missing right -> use len(s) [far right of string]

s = “Computer”
print(s[1:])  # prints omputer
print(s[:5])  # prints Compu
print(s[-2:])  # prints er

Practice

• Write a function called total_seconds that takes one string argument.
  This argument will be a string of the form "M:SS" where M is a number of
  minutes (a single digit) and SS is a number of seconds (2 digits). This
  function should calculate the total number of seconds in this amount of
  time and return it as an integer. (Hint: Use string slicing/indices)

• Write a function called count_digits that returns the number of digits
  in a string.
  – count_digits("abc123def5") returns 4

• Write a function called sum_digits that returns the sum of all the digits in
  a string.
  – sum_digits("abc123def5") returns 11
    (because 1 + 2 + 3 + 5 = 11)
String Concatenation

- Combines two strings into a new, longer string
- Uses the same plus sign as addition

```python
s1 = "CS141"
s2 = "rocks!"
bigstring = s1 + s2
print(bigstring) #prints CS141rocks!
```

String Concatenation

- Unlike print(), string concatenation does not put spaces between your strings.

```python
s1 = "CS141"
s2 = "rocks!"
bigstring = s1 + " " + s2
print(bigstring) #prints CS141 rocks!
```

Sample Problem

- All professor email addresses at Rhodes are constructed from the professor's last name, followed by the first initial of their first name.

- We want to design a function that take's a prof's first and last name and returns their email address.

Sample Problem Solution

```python
def make_prof_email(first, last):
    init = first[0]
    address = last + init + "@rhodes.edu"
    return address

def main():
    firstname = input("First name: ")
    lastname = input("Last name: ")
    addr = make_prof_email(firstname, lastname)
    print("Email": addr)
```
The Repetition Operator

- **Repetition operator**: makes multiple copies of a string and joins them together
  - The * symbol is a repetition operator when applied to a string and an integer
    - String is left operand; number is right
    - General format: string_to_copy * n
  - Variable references a new string which contains multiple copies of the original string

  ```
s = "a"
s2 = s * 10
print(s2)   #Output is aaaaaaaaaa
```

Other String Methods

- Programs commonly need to search for substrings
- Several methods to accomplish this:
  - `endswith(substring)` checks if the string ends with substring
  - Returns True or False
  - `startswith(substring)` checks if the string starts with substring
  - Returns True or False

More String Methods

- Several methods to accomplish this (cont'd):
  - `find(substring)` searches for substring within the string
    - Returns lowest index of the substring, or if the substring is not contained in the string, returns -1
  - `replace(substring, new_string)`:
    - Returns a copy of the string where every occurrence of substring is replaced with new_string

Using the `find` method

```python
def main():
    filename = "first_last_assignsubmission_file_lastname_firstname_prg6.py"
    print(renamefile(filename))
def renamefile(filename):
    ind = filename.find("file ")
    filename = filename[ind+1:]
    return filename
main()
```

Output:
`lastnameFirstname_prg6.py`
String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>endswith(substring)</td>
<td>The substring argument is a string. The method returns true if the string ends with substring.</td>
</tr>
<tr>
<td>find(substring)</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>replace(old, new)</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>startswith(substring)</td>
<td>The substring argument is a string. The method returns true if the string starts with substring.</td>
</tr>
</tbody>
</table>

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

Practice

- Write a function that returns a Rhodes student email address. (Assume this email address is for a new student). Your function will need to take in 4 arguments: first name, last name, middle name and class year.
- Write a function called reverse that takes a string argument and returns the string argument with all characters in the reverse order.
  - `reverse("Welsh")` returns "hsleW"
- Write a function called filter_digits that returns only the digits from a string.
  - `filter_digits("abc123def5")` returns "1235"
- 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