CSCI 141

Lecture 16
String Manipulation
Announcements
Announcements

• Midterm grades are out, along with a long announcement with many details. Among them:
Announcements

• Midterm grades are out, along with a long announcement with many details. Among them:
  • Review your exam on Gradescope for 2 bonus points
Announcements

- Midterm grades are out, along with a long announcement with many details. Among them:
  - Review your exam on Gradescope for 2 bonus points
  - Grades are curved
Announcements

• Midterm grades are out, along with a long announcement with many details. Among them:

  • Review your exam on Gradescope for 2 bonus points

  • Grades are curved

  • If you do better on the final, it will replace your midterm grade.
Announcements

• Midterm grades are out, along with a long announcement with many details. Among them:
  
  • Review your exam on Gradescope for 2 bonus points
  
  • Grades are curved
  
  • If you do better on the final, it will replace your midterm grade.

• A4 is due Friday.
Announcements

• Midterm grades are out, along with a long announcement with many details. Among them:
  • Review your exam on Gradescope for 2 bonus points
  • Grades are curved
  • If you do better on the final, it will replace your midterm grade.

• A4 is due Friday.
  • I’ve updated the rubric - it’s now worth 80 points.
Announcements

• Midterm grades are out, along with a long announcement with many details. Among them:
  • Review your exam on Gradescope for 2 bonus points
  • Grades are curved
  • If you do better on the final, it will replace your midterm grade.

• A4 is due Friday.
  • I’ve updated the rubric - it’s now worth 80 points.
  • If you haven’t started yet, start now.
Goals

• Review what we know already about strings:
  • the str type, + and * operators, len function
Last time...

- Returning from functions
- Using functions to wrap up complex things
- Function definition order
- Tuples:
  - packing, unpacking via the assignment operator
  - as return values and as parameters
A new data type: tuples

• A tuple is a sequence of values, optionally enclosed in parens.

  (1, 4, "Mufasa")

• You can “pack” and “unpack” them using assignment statements:

  v = (1, 4, "Mufasa") # "packing"

  (a, b, c) = v # "unpacking"
Docstrings, Preconditions and Postconditions

• Every function should have a docstring describing its behavior.

• When applicable, a docstring should include:
  
  • **Preconditions**: any assumptions the function must make to work.

  • **Postconditions**: things that are guaranteed to be true after the function finishes executing.
Reminder: Docstrings, Preconditions and Postconditions

**Example.** Suppose you wrote this function:

```python
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a restaurant bill of bill_amt, assuming a tip percent of tip_pct and splitting the bill evenly among num_diners people. """
    
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```
Example. Suppose you wrote this function:

def split_bill(bill_amt, tip_pct, num_diners):
    
    """ Return the total owed by each diner for a restaurant bill of bill_amt, assuming a tip percent of tip_pct and splitting the bill evenly among num_diners people. 
    """

    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
Reminder: Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```python
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a
    restaurant bill of bill_amt, assuming a tip
    percent of tip_pct and splitting the bill
    evenly among num_diners people.
    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners

>>> split_bill(34.78, 18.0, 0)
```

```python
3.92
```
Reminder: Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```python
def split_bill(bill_amt, tip_pct, num_diners):
    
    """ Return the total owed by each diner for a restaurant bill of bill_amt, assuming a tip percent of tip_pct and splitting the bill evenly among num_diners people. """

    
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners

>>> split_bill(34.78, 18.0, 0)
ZeroDivisionError: float division by zero
```
Reminder: Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a restaurant bill of bill_amt, assuming a tip percent of tip_pct and splitting the bill evenly among num_diners people."
    
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners

>>> split_bill(34.78, 18.0, 0)
ZeroDivisionError: float division by zero

Bad news:
Reminder: Docstrings, Preconditions and Postconditions

**Example.** Suppose you wrote this function:

```python
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a restaurant bill of bill_amt, assuming a tip percent of tip_pct and splitting the bill evenly among num_diners people. """

    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners

>>> split_bill(34.78, 18.0, 0)
ZeroDivisionError: float division by zero

Bad news: This is your fault.
```
Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```python
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a restaurant bill of bill_amt, assuming a tip percent of tip_pct and splitting the bill evenly among num_diners people.

    Precondition: num_diners > 0
    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```
Example. Suppose you wrote this function:

```python
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a restaurant bill of bill_amt, assuming a tip percent of tip_pct and splitting the bill evenly among num_diners people.
    ""
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

Precondition: num_diners > 0
Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```python
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a restaurant bill of bill_amt, assuming a tip percent of tip_pct and splitting the bill evenly among num_diners people.
    """

    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners

>>> split_bill(34.78, 18.0, 0)
```
Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```python
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a restaurant bill of bill_amt, assuming a tip percent of tip_pct and splitting the bill evenly among num_diners people.
    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

```python
>>> split_bill(34.78, 18.0, 0)
ZeroDivisionError: float division by zero
```
Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```python
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a restaurant bill of bill_amt, assuming a tip percent of tip_pct and splitting the bill evenly among num_diners people.
    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

```python
>>> split_bill(34.78, 18.0, 0)
ZeroDivisionError: float division by zero
```

This is my fault.
Tuples are sequences, so they can be used in for loops just like lists and ranges.

These two loops do the same thing:

```python
for number in [1, 3]:
    print(number, ">", sep="<", end="")

for number in (1, 3):
    print(number, ">", sep="<", end="")
```
Tuples are sequences, so they can be used in for loops just like lists and ranges.

These two loops do the same thing:

```python
for number in [1, 3]:
    print(number, '>', sep='<', end='')

for number in (1, 3):
    print(number, '>', sep='<', end='')
```

What do they print?

A. <1> <3>  
B. 1<>3  
C. 1<>3<>  
D. 1<<3>
Today’s Quiz

• 3 minutes
Today’s Quiz

• 3 minutes

• Working with a neighbor: do your answers agree? (2 minutes)
Today: Strings

Don’t we already know about strings?
Today: Strings

Don’t we already know about strings?

`type("hello")`
Today: Strings

Don’t we already know about strings?

```python
type("hello")  # => <class 'str'>
```
Today: Strings

Don’t we already know about strings?

```
> type("hello")  # => <class 'str'>
> print("Hello")
```
Today: Strings

Don’t we already know about strings?

```python
type("hello")  # => <class ‘str’>

print("Hello")  # prints Hello to the console
```
Today: Strings

Don’t we already know about strings?

def type("hello")  # => <class 'str'>
def print("Hello")  # prints Hello to the console

def "Hello" + "World"
Today: Strings

Don’t we already know about strings?

```python
print("Hello")  # prints Hello to the console

"Hello" + "World"  # => "HelloWorld"
```
Today: Strings

Don’t we already know about strings?

```python
type("hello")  # => <class 'str'>

print("Hello")  # prints Hello to the console

"Hello" + "World"  # => "HelloWorld"

len("abc")
```
Today: Strings

Don’t we already know about strings?

```python
type("hello")  # => <class ‘str’>

print("Hello")  # prints Hello to the console

"Hello" + "World"  # => “HelloWorld”

len("abc")  # => 3
```
Today: Strings

Don’t we already know about strings?

```python
type("hello")  # => <class 'str'>

print("Hello")  # prints Hello to the console

"Hello" + "World"  # => "HelloWorld"

len("abc")  # => 3

"na" * 16 + " Batman!"
```
Today: Strings

Don’t we already know about strings?

```python
import string

# => <class 'str'>
print(type("hello"))

# prints Hello to the console
print("Hello")

# => "HelloWorld"
print("Hello" + "World")

# => 3
print(len("abc"))

# => "na" * 16 + " Batman!"
print("na" * 16 + " Batman!")
```
Today: Strings

Don’t we already know about strings?

```python
type("hello")  # => <class ‘str’>

print("Hello")  # prints Hello to the console

"Hello" + "World"  # => “HelloWorld”

len("abc")  # => 3

"na" * 16 + " Batman!"
# => … "nananananananananananananana Batman!"
```
Strings: What else is there?
def house_number(address_line):
Strings: What else is there?

def house_number(address_line):
    """ Return the house number portion of
"""
def house_number(address_line):
    """ Return the house number portion of
    the given address line. """
def house_number(address_line):
    """ Return the house number portion of the given address line.
    Examples:
Strings: What else is there?

def house_number(address_line):
    """ Return the house number portion of the given address line.
    Examples:
        house_number("1600 Pennsylvania Ave")
def house_number(address_line):
    """ Return the house number portion of
    the given address line.
    Examples:
    house_number("1600 Pennsylvania Ave")
    => 1600
def house_number(address_line):
    """ Return the house number portion of
    the given address line.
    Examples:
    house_number("1600 Pennsylvania Ave")
    => 1600
    house_number("221B Baker St")
def house_number(address_line):
    """ Return the house number portion of
    the given address line.
    Examples:
    house_number("1600 Pennsylvania Ave")
    => 1600
    house_number("221B Baker St")
    => 221
def house_number(address_line):
    """ Return the house number portion of the given address line.
    Examples:
    house_number("1600 Pennsylvania Ave")
    => 1600
    house_number("221B Baker St")
    => 221
    """
def house_number(address_line):
    """ Return the house number portion of the given address line.
    Examples:
    house_number("1600 Pennsylvania Ave")
    => 1600
    house_number("221B Baker St")
    => 221
    """
def house_number(address_line):
    """ Return the house number portion of 
    the given address line. 
    Examples:
    house_number("1600 Pennsylvania Ave")
        => 1600
    house_number("221B Baker St")
        => 221
    """

    # ????
    return result
Strings: What else is there?

def house_number(address_line):
    """ Return the house number portion of the given address line.
    Examples:
        house_number("1600 Pennsylvania Ave")
        => 1600
        house_number("221B Baker St")
        => 221
    """

    # ????
    return result
Strings: What else is there?
def ignore_comments(line_of_code):

Strings: What else is there?
Strings: What else is there?

def ignore_comments(line_of_code):
    """ Return a line of code with any comments"""
Strings: What else is there?

def ignore_comments(line_of_code):
    """ Return a line of code with any comments after a # sign removed."""
def ignore_comments(line_of_code):
    """ Return a line of code with any comments after a # sign removed. """
def ignore_comments(line_of_code):
    """ Return a line of code with any comments after a # sign removed.
    """
    # ????
def ignore_comments(line_of_code):
    """ Return a line of code with any comments
    after a # sign removed.
    """

    # ????

    return result
def ignore_comments(line_of_code):
    """ Return a line of code with any comments after a # sign removed. """
    # ????
    return result
Strings are sequences, so they can be used in for loops just like lists and ranges.

Check this out:

```python
for letter in "Bellingham":
    print(letter, "-", sep=" ", end=
```

"""
```
Strings are sequences, so they can be used in for loops just like lists and ranges.

Check this out:

```python
for letter in "Bellingham":
    print(letter, "-", sep=" ", end=" ")
```

What does this print?

- A. Bellingham
- B. B-e-l-l-i-n-g-h-a-m
- C. -B-e-l-l-i-n-g-h-a-m
- D. B-e-l-l-i-n-g-h-a-m-
Exercise (not collected)

Write a function that **prints** a string with all vowels removed.

```python
def remove_vowels(string):
    """ Print string, but with no vowels. 
    Don't count y as a vowel. """
```

**Modification**: Return the modified string instead of printing it.
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters. We can get access to an individual character by index.

outlook = “Summer is near”

How is this stored in memory?
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters. We can get access to an individual character by index.

outlook = "Summer is near"

How is this stored in memory?
Strings are collections of individual characters. We can get access to an individual character by index.

`outlook = “Summer is near”`

How is this stored in memory?

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Summer</td>
</tr>
<tr>
<td>1</td>
<td>is</td>
</tr>
<tr>
<td>2</td>
<td>near</td>
</tr>
</tbody>
</table>
Strings are collections of individual characters. We can get access to an individual character by index.

```
outlook = "Summer is near"
```

How is this stored in memory?

```
<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>S</td>
</tr>
<tr>
<td>1</td>
<td>u</td>
</tr>
<tr>
<td>2</td>
<td>m</td>
</tr>
<tr>
<td>3</td>
<td>m</td>
</tr>
<tr>
<td>4</td>
<td>e</td>
</tr>
<tr>
<td>5</td>
<td>r</td>
</tr>
<tr>
<td>6</td>
<td>i</td>
</tr>
<tr>
<td>7</td>
<td>s</td>
</tr>
<tr>
<td>8</td>
<td>is</td>
</tr>
<tr>
<td>9</td>
<td>near</td>
</tr>
</tbody>
</table>
```

Indices in Python begin at 0.
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters. We can get access to an individual character by index.

outlook = “Summer is near”

How is this stored in memory?

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Index:</th>
<th>Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>outlook</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13</td>
<td>Summer is near</td>
</tr>
</tbody>
</table>

Indices in Python begin at 0.

Syntax:
Indexing into Strings

Strings are collections of individual characters. We can get access to an individual character by index.

outlook = "Summer is near"

How is this stored in memory?

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;S&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;u&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;m&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;m&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;e&quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot;r&quot;</td>
</tr>
<tr>
<td>6</td>
<td>&quot;i&quot;</td>
</tr>
<tr>
<td>7</td>
<td>&quot;s&quot;</td>
</tr>
<tr>
<td>8</td>
<td>&quot;i&quot;</td>
</tr>
<tr>
<td>9</td>
<td>&quot;n&quot;</td>
</tr>
<tr>
<td>10</td>
<td>&quot;e&quot;</td>
</tr>
<tr>
<td>11</td>
<td>&quot;r&quot;</td>
</tr>
</tbody>
</table>

Syntax:

outlook[0] # => "S"

Indices in Python begin at 0.
Indexing into Strings

Strings are collections of individual characters. We can get access to an individual character by index.

outlook = "Summer is near"

How is this stored in memory?

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Summer</td>
</tr>
<tr>
<td>1</td>
<td>is</td>
</tr>
<tr>
<td>2</td>
<td>near</td>
</tr>
</tbody>
</table>

Syntax:

outlook[0] # => "S"
outlook[4] # => "e"

Indices in Python begin at 0.
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters. We can get access to an individual character by index.

outlook = "Summer is near"

How is this stored in memory?

Indices in Python begin at 0.

Spaces are characters too!

Syntax:

```
outlook[0] # => "S"
outlook[4] # => "e"
```
Indexing into Strings
(just smaller strings!)

Strings are collections of individual characters. We can get access to an individual character by index.

```
outlook = "Summer is near"
```

How is this stored in memory?

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Outlook</td>
</tr>
<tr>
<td>1</td>
<td>&quot;S&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;u&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;m&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;m&quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot;e&quot;</td>
</tr>
<tr>
<td>6</td>
<td>&quot;r&quot;</td>
</tr>
<tr>
<td>7</td>
<td>&quot;r&quot;</td>
</tr>
<tr>
<td>8</td>
<td>&quot;i&quot;</td>
</tr>
<tr>
<td>9</td>
<td>&quot;s&quot;</td>
</tr>
<tr>
<td>10</td>
<td>&quot;n&quot;</td>
</tr>
<tr>
<td>11</td>
<td>&quot;e&quot;</td>
</tr>
<tr>
<td>12</td>
<td>&quot;a&quot;</td>
</tr>
<tr>
<td>13</td>
<td>&quot;r&quot;</td>
</tr>
</tbody>
</table>

Syntax:

```
outlook[0]  # => "S"
outlook[4]  # => "e"
```

Indices in Python begin at 0.

Spaces are characters too!

```
outlook[6]  # => "  
```
Indexing into Strings

Strings are collections of individual characters. We can get access to an individual character by index.

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>S</td>
<td>u</td>
<td>m</td>
<td>m</td>
<td>e</td>
<td>r</td>
<td>i</td>
<td>s</td>
<td>n</td>
<td>e</td>
<td>a</td>
<td>r</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Problem**: Return a string with any text after and including the # symbol removed.

```python
def remove_comments(string):
    """ Remove all characters starting with a # symbol from string, and return the result. """
```
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters. We can get access to an individual character by index.

<table>
<thead>
<tr>
<th>Index:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value:</td>
<td>S</td>
<td>u</td>
<td>m</td>
<td>m</td>
<td>e</td>
<td>r</td>
<td>i</td>
<td>s</td>
<td>n</td>
<td>e</td>
<td>a</td>
<td>r</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Problem:** Return a string with any text after and including the # symbol removed.

```python
def remove_comments(string):
    """ Remove all characters starting with a # symbol from string, and return the result. """
```
Indexing into Strings

Index:  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13  
Value: | S | u | m | m | e | r | i | s | n | e | a | r |    

ABCD: What is the index of the last character of a string $s$?

A. $\text{len}(s)$  
B. $\text{len}(s - 1)$  
C. $\text{len}(s + 1)$  
D. 42
A4 (Revisited, briefly)

Your task:
Draw this.
A4 (Revisited, briefly)

Your task: Draw this.

Sounds simple, right?
Your task:
Draw this.

Sounds simple, right?

No.
A4: Pseudocode

# Let p be a random point in the window
# loop 10000 times:
#   c = a random corner of the triangle
#   m = the midpoint between p and c
#   choose a color for m
#   color the pixel at m
#   p=m
# Let p be a random point in the window
# loop 10000 times:
#    c = a random corner of the triangle
#    m = the midpoint between p and c
#    choose a color for m
#    color the pixel at m
#    p=m
# Let p be a random point in the window
# loop 10000 times:
#     c = a random corner of the triangle
#     m = the midpoint between p and c
#     choose a color for m
#     color the pixel at m
#     p=m

Demo:
• solution in action
• making up function names