Using Objects and Images in Python

Look at little UTA Tiffany using objects! Learn from her!

*based in part on notes from the CS-for-All curriculum developed at Harvey Mudd College*
What Is An Object?

• An object is a construct that groups together:
  • one or more data values (the object's attributes)
  • one or more functions that operate on those data values (known as the object's methods)

• Objects are typically nouns
  • Attributes correspond to adjectives (i.e., properties of the noun)
  • Methods correspond to verbs that act on the noun
Strings Are Objects

• In Python, a string is an object.
  • **attributes:**
    • the characters in the string
    • the length of the string
  • **methods:** functions inside the string that we can use to operate on the string

<table>
<thead>
<tr>
<th>Method</th>
<th>String for 'hello'</th>
<th>String for 'bye'</th>
</tr>
</thead>
<tbody>
<tr>
<td>contents</td>
<td>'h', 'e', 'l', 'l', 'o'</td>
<td>'b', 'y', 'e'</td>
</tr>
<tr>
<td>length</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>upper()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lower()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>find()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>count()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>replace()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>split()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Calling a Method

• An object's methods are inside the object, so we use *dot notation* to call them.

• Example:

```python
name = 'Perry'
allcaps = name.upper()
```

• Because a method is inside the object, it is able to access the object's attributes.
String Methods (partial list)

- `s.lower()` return a copy of `s` with all lowercase characters
- `s.upper()` return a copy of `s` with all uppercase characters
- `s.find(sub)` return the index of the first occurrence of the substring `sub` in the string `s` (-1 if not found)
- `s.count(sub)` return the number of occurrences of the substring `sub` in the string `s` (0 if not found)
- `s.replace(target, repl)` replace all occurrences of the substring `target` in `s` with the substring `repl`
Examples of Using String Methods

```python
>>> weather = 'A snowy start to Spring!

>>> weather.upper()
'A SNOWY START TO SPRING!'

>>> weather.lower()
'a snowy start to spring'

>>> weather.replace('s', 'f')
'A fnowy ftart to Spring!

>>> weather
'A snowy start to Spring!'
```
Splitting a String

• The `split()` method breaks a string into a list of substrings.

```
>>> name = 'Martin Luther King'
>>> name.split()
['Martin', 'Luther', 'King']
>>> components = name.split()
>>> components[0]
'Martin'
```

• By default, it uses *whitespace characters* (spaces, tabs, and newlines) to determine where the splits should occur.

• You can specify a different separator:

```
>>> date = '11/10/2014'
>>> date.split('/
')['11', '10', '2014']
```
Each Image object has:

- **attributes:**
  - the name of the image
  - the height of the image
  - the width of the image
  - the pixels in the image

- **methods:**
  - `img.get_height()` – returns the height of the image `img`
  - `img.get_width()` – returns the width of the image `img`
  - `img.get_pixel(r, c)` – returns the list of RGB values for the pixel at position `(r, c)` in the image `img`
  - `img.set_pixel(r, c, rgb)` – changes the RGB values for the pixel at position `(r, c)` in `img` to the list `rgb`
Different Image Objects for Different Images

**image**

![Image of SPAM can]

**Image object**

<table>
<thead>
<tr>
<th>name</th>
<th>'spam.png'</th>
</tr>
</thead>
<tbody>
<tr>
<td>height</td>
<td>334</td>
</tr>
<tr>
<td>width</td>
<td>338</td>
</tr>
<tr>
<td>pixels</td>
<td>a list of lists</td>
</tr>
<tr>
<td>get_height</td>
<td>get_pixel</td>
</tr>
<tr>
<td>get_width</td>
<td>set_pixel</td>
</tr>
</tbody>
</table>

![Image of cartoon dog]

<table>
<thead>
<tr>
<th>name</th>
<th>'rhett.png'</th>
</tr>
</thead>
<tbody>
<tr>
<td>height</td>
<td>420</td>
</tr>
<tr>
<td>width</td>
<td>274</td>
</tr>
<tr>
<td>pixels</td>
<td>a list of lists</td>
</tr>
<tr>
<td>get_height</td>
<td>get_pixel</td>
</tr>
<tr>
<td>get_width</td>
<td>set_pixel</td>
</tr>
</tbody>
</table>
The color of each pixel is represented by a list of 3 integers:

\[\text{[red, green, blue]}\]

- example: the pink pixel at right has color
  \[\text{[240, 60, 225]}\]
- known as RGB values
- each value is between 0-255

Other examples:
- pure red: \[\text{[255, 0, 0]}\]
- pure green: \[\text{[0, 255, 0]}\]
- pure blue: \[\text{[0, 0, 255]}\]
- white: \[\text{[255, 255, 255]}\]
- black: \[\text{[0, 0, 0]}\]
Nested loops and 
2D structure

```python
for r in range(h):
    for c in range(w):
        # process the pixel at (r, c)
```
The frequency of (leaked) 4-digit PIN codes. Brighter color reflects higher frequency. The brightness in the lower left corner reflects people choosing their birth month (1 – 12) and day (1 – 31); the vertical line suggests birth years (i.e., the first two digits are 19); the diagonal line reflects a preference for repeated couplets of numbers (e.g., 1212 or 3636).
The menu to implement:

(0) Input a new list of prices
(1) Print the current list
(2) Find the latest price
(3) Find the average price
(4) Find the standard deviation
(5) Find the min and its day
(6) Find the max and its day
(7) Test a threshold
(8) Your TTS investment plan
(9) Quit

Enter your choice:

hw02: T.T. Securities (TTS)

Analyzes a sequence of stock prices

prices = [45, 80, 10, 30, 27, 50, 5, 15]
Our starter code

def display_menu():
    """ prints a menu of options
    """

    print()
    print('(0) Input a new list of prices')
    print('(1) Print the current prices')
    print('(2) Find the latest price')
    ...
    print('(9) Quit')
    print()
def tts():
    prices = []
    while True:
        display_menu()
        choice = int(input('Enter your choice: '))
        print()
        if choice == 0:
            prices = get_new_prices()
        elif choice == 9:
            break
        elif choice == 1:
            print_prices(prices)
        elif choice == 2:
            latest = latest_price(prices)
            print('The latest price is', latest)
        ## add code to process the other choices here
        ...
    print('See you yesterday!')
User Input

- Getting a *string value* from the user:
  
  ```python
  variable = input(prompt)
  ```
  
- Getting an *integer value*:
  
  ```python
  variable = int(input(prompt))
  ```

- Getting a *floating-point value*:
  
  ```python
  variable = float(input(prompt))
  ```

- Getting an arbitrary non-string value (e.g., a list):
  
  ```python
  variable = eval(input(prompt))
  ```
  
  - eval treats a string as an expression to be evaluated

- Examples:
  
  ```python
  name = input('name of assignment: ')
count = int(input('possible points: '))
scores = eval(input('list of scores: '))
  ```
User Input

• Getting a *string value* from the user:

```python
variable = input(prompt)
```
where `prompt` is a string

• Getting an *integer value*:

```python
variable = int(input(prompt))
```

• Getting a *floating-point value*:

```python
variable = float(input(prompt))
```

• Examples:

```python
name = input('name of assignment: ') 
count = int(input('possible points: ')) 
price = float(input('enter a price: '))
```
def get_new_prices():
    new_list = eval(input('Enter new prices: '))
    return new_list

def print_prices(prices):
    """ prints the current list of prices
    input: prices is a list of 1 or more numbers.
    """
    ## IMPORTANT: You will need to change this...
    print('current prices:', prices)

def latest_price(prices):
    return prices[-1]
Our starter code

def get_new_prices():
    ''' gets a new list of prices from the user and returns it '''
    try:
        new_price_list = input("Enter a new list of prices: ")
        new_price_list = [float(x) for x in \new_price_list.split(' ')]
        return new_price_list
    except:
        print('\nInvalid input. System exiting...\n')
        exit()

def print_prices(prices):
    ''' prints the current list of prices 
    input: prices is a list of 1 or more numbers. 
    '''
    ## IMPORTANT: You will need to change this...
    print('current prices:', prices)

def latest_price(prices):
    return prices[-1]
Functions you'll write

All use loops...

Menu

(0) Input a new list of prices
(1) Print the current list
(2) Find the latest price
(3) Find the average price
(4) Find the standard deviation
(5) Find the min and its day
(6) Find the max and its day
(7) Test a threshold
(8) Your TTS investment plan
(9) Quit
Enter your choice:

\[
\sqrt{\frac{\sum_{i} (L[i] - L_{av})^2}{\text{len}(L)}}
\]
Min price

What's the *idea* for finding the smallest (minimum) price?

\[ L = [45, 80, 10, 30, 27, 50, 5, 15] \]

\[ m = \]

\( m \) is the "min so far"

track the value of the *minimum so far* as you loop over list
```python
def minprice(prices):
    m = prices[0]
    for x in prices:
        if x < m:
            m = x
    return m

L = [45, 80, 10, 30, 27, 50, 5, 15]
```

Min price

\[ L = [45, 80, 10, 30, 27, 50, 5, 15] \]
def minprice(prices):
    m = prices[0]
    for x in prices:
        if x < m:
            m = x
    return m

What about the \textit{day} of the minimum price?
T.T. Securities
==
*Time Travel*
Securities!

(0) Input a new list of prices
(1) Print the current list
(2) Find the latest price
(3) Find the average price
(4) Find the standard deviation
(5) Find the min and its day
(6) Find the max and its day
(7) Test a threshold
(8) Your TTS investment plan
(9) Quit

Enter your choice:
def minday(prices):
    for i in range(len(prices)):  # index-based!
        if _________:
            return mi

L = [45, 80, 10, 30, 27, 50, 5, 15]

6 should be returned
The TTS Advantage!

Your stock's prices:  \[ L = [ 45, 80, 10, 30, 27, 50, 5, 15 ] \]

<table>
<thead>
<tr>
<th>Day</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>45.00</td>
</tr>
<tr>
<td>1</td>
<td>80.00</td>
</tr>
<tr>
<td>2</td>
<td>10.00</td>
</tr>
<tr>
<td>3</td>
<td>30.00</td>
</tr>
<tr>
<td>4</td>
<td>27.00</td>
</tr>
<tr>
<td>5</td>
<td>50.00</td>
</tr>
<tr>
<td>6</td>
<td>5.00</td>
</tr>
<tr>
<td>7</td>
<td>15.00</td>
</tr>
</tbody>
</table>

What is the best TTS investment strategy here?

You may only sell \textbf{after} you buy.
The TTS Advantage!

Your stock's prices:  \( L = [ 45, 80, 10, 30, 27, 50, 5, 15 ] \)

<table>
<thead>
<tr>
<th>Day</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>45.00</td>
</tr>
<tr>
<td>1</td>
<td>80.00</td>
</tr>
<tr>
<td>2</td>
<td>10.00</td>
</tr>
<tr>
<td>3</td>
<td>30.00</td>
</tr>
<tr>
<td>4</td>
<td>27.00</td>
</tr>
<tr>
<td>5</td>
<td>50.00</td>
</tr>
<tr>
<td>6</td>
<td>5.00</td>
</tr>
<tr>
<td>7</td>
<td>15.00</td>
</tr>
</tbody>
</table>

What is the best TTS investment strategy here?

You may only sell \textit{after} you buy.
Finding a minimum difference

```python
def diff(l1, l2):
    # Hint! Track the smallest absolute diff. between any value from l1 and any value from l2.

    # Hint! Use nested loops!

    # Hint! Track the min diff so far as you loop over l1 and l2...
```

```python
>>> diff([12, 3, 7], [6, 0, 5])
1
```
Which of these works?

A.
```python
def diff(l1, l2):
    mindiff = abs(l1[0] - l2[0])
    for x in l1:
        for y in l2:
            d = abs(x - y)
            if d < mindiff:
                mindiff = d
    return mindiff
```

B.
```python
def diff(l1, l2):
    mindiff = 0
    for x in l1:
        for y in l2:
            d = abs(x - y)
            if d < mindiff:
                mindiff = d
    return mindiff
```

C.
```python
def diff(l1, l2):
    mindiff = abs(l1[0] - l2[0])
    for x in l1:
        for y in l2:
            d = abs(x - y)
            if d < mindiff:
                return d
            else:
                return mindiff
```

D. more than one of them
Which of these works?

A. 
```python
def diff(l1, l2):
    mindiff = abs(l1[0] - l2[0])
    for x in l1:
        for y in l2:
            d = abs(x - y)
            if d < mindiff:
                mindiff = d
    return mindiff
```

B. 
```python
def diff(l1, l2):
    mindiff = 0
    for x in l1:
        for y in l2:
            d = abs(x - y)
            if d < mindiff:
                mindiff = d
    return mindiff
```

C. 
```python
def diff(l1, l2):
    mindiff = abs(l1[0] - l2[0])
    for x in l1:
        for y in l2:
            d = abs(x - y)
            if d < mindiff:
                return d
            mindiff = d
    return mindiff
```

D. more than one of them
What if we want the indices of the min-diff values?

```python
def diff_indices(l1, l2):
    # what needs to change?
    mindiff = abs(l1[0] - l2[0])

    for x in l1:
        for y in l2:
            d = abs(x - y)
            if d < mindiff:
                mindiff = d

    return mindiff
```

```python
>>> diff_indices([12,3,7], [6,0,5])

position 2 in first list
position 0 in second list
```

should `print` instead of returning
What if we want the indices of the min-diff values?

```python
def diff_indices(l1, l2):
    mindiff = abs(l1[0] - l2[0])
    pos1 = 0
    pos2 = 0

    for i in range(len(l1)):
        for j in range(len(l2)):
            d = abs(l1[i] - l2[j])
            if d < mindiff:
                mindiff = d
                pos1 = i
                pos2 = j
    print('position', pos1, 'in first list')
    print('position', pos2, 'in second list')
```

```bash
>>> diff_indices([12,3,7], [6,0,5])
position 2 in first list
position 0 in second list
```

should print instead of returning