Classes + Objects

Chris Gregg

Based on Slides by Chris Piech and Mehran Sahami

CS106A, Stanford University
The Stanford Honor Code
- CS106A retraction policy
- Deadline to retract any assignments: August 10th
Learning Goals

1. Learning about Object-Oriented Programming
2. Writing code using Classes and Objects in Python
Object-Oriented Programming (OOP)
It's not a mistake!
Object-Oriented Programming

• There are different paradigms in programming

• So far, you've learned imperative programming
  – Provide series of direct commands for program execution
  – Commands are changing the program's state

• Object-oriented programming
  – Define objects that contain data and behavior (functions)
  – Program is (mostly) an interaction between objects
  – You are calling function of objects (called "methods")

• Python allows for programming in either paradigm!
  – Other programming paradigms exist, but we won't talk about those in this class
What are Classes and Objects?

• Classes are like blueprints
  – They provide a template for a kind of object
  – They define a new type
  – E.g., "Human" would be a class
    • Generally, have 2 arms, have two legs, breathe air, etc.

• Objects are instances of Classes
  – Can have multiple objects of the same Class type
  – E.g., You would be an instance of the Human class
    • So, you have the properties of your Class (Human)
  – There are lots of other people out there too
    • You are all of type "Human"
    • You are all objects of the same Class
Example of a Class in Python

• Let's create a Counter class
  – Can ask is for the "next" ticket number
  – Need to keep track of next ticket number
  – Class names start with **Uppercase** character
  – No `main()` function (Class is **not** a program)

```python
class Counter:

    # Constructor
    def __init__(self):
        self.ticket_num = 0   # "instance" variable

    # Method (function) that returns next ticket value
    def next_value(self):
        self.ticket_num += 1
        return self.ticket_num
```
Let's See It In Action: counter.py
Objects are Mutable

• When you pass an object as a parameter, changes to object in that function persist after function ends

```python
from counter import Counter  # import the Class

def count_two_times(count):
    for i in range(2):
        print(count.next_value())

def main():
    count1 = Counter()
    count2 = Counter()
    print('Count1: ')
    count_two_times(count1)
    print('Count2: ')
    count_two_times(count2)
    print('Count1: ')
    count_two_times(count1)

output:
Count1: 1 2
Count2: 1 2
Count1: 3 4
```
General Form for Writing a Class

• Filename for class is usually `classname.py`
  – Filename is usually lowercase version of class name in file

```python
class Classname:

    # Constructor
    def __init__(self, additional parameters):
        body
        self.variable_name = value  # example instance variable

    # Method
    def method_name(self, additional parameters):
        body
```
Constructor of a Class

• Constructor
  – Syntax:
    ```python
def __init__(self, additional parameters):
    body
```

• Called when a new object is being created
  – Does not explicitly specify a return value
  – New object is created and returned
    • Can think of constructor as the "factory" that creates new objects
    – Responsible for initializing object (setting initial values)
    – Generally, where instance variables are created (with `self`)
      ```python
      self.variable_name = value  # create instance variable
      ```
Instance Variables

- Instance variables are variable associated with objects
  - Each object get its **own set** of instance variables
  - Generally, they are initialized in constructor for class
  - Instance variables accessed using **self**:
    ```python
    self.variable_name = value
    ```
  - Self really refers to the object that a method is called on

```python
def main():
    count1 = Counter()
    count2 = Counter()
    x = count1.next_value()
    y = count2.next_value()
```
Instance Variables

• Instance variables are variable associated with objects
  – Each object gets its **own set** of instance variables
  – Generally, they are initialized in constructor for class
  – Instance variables accessed using **self**:
    
    ```
    self.\texttt{variable\_name} = \texttt{value}
    ```
  – Self really refers to the object that a method is called on

```python
def main():
    count1 = Counter()
    count2 = Counter()
    x = count1.next_value()
    y = count2.next_value()

def \_\_init\_(self):
    self.ticket\_num = 0

```

```python
count1 \rightarrow
    self.ticket\_num \quad 0
```
Instance Variables

- Instance variables are variable associated with objects
  - Each object get its own set of instance variables
  - Generally, they are initialized in constructor for class
  - Instance variables accessed using self:
    
    ```python
    self.\texttt{variable\_name} = \texttt{value}
    ```
  - Self really refers to the object that a method is called on

```python
def main():
    count1 = Counter()
    count2 = Counter()
    x = count1.next_value()
    y = count2.next_value()
```

```python
count1 \rightarrow \begin{array}{c}
  \texttt{self.ticket\_num} \\
  0
\end{array}
```
• Instance variables are variable associated with objects
  – Each object get its own set of instance variables
  – Generally, they are initialized in constructor for class
  – Instance variables accessed using self:
    
    
    ```python
    self.variable_name = value
    ```
  – Self really refers to the object that a method is called on

```python
def main():
    count1 = Counter()
    count2 = Counter()
    x = count1.next_value()
    y = count2.next_value()
```

```python
def __init__(self):
    self.ticket_num = 0
```
Instance Variables

- Instance variables are variable associated with objects
  - Each object gets its own set of instance variables
  - Generally, they are initialized in constructor for class
  - Instance variables accessed using `self`:
    ```python
    self.\texttt{variable\_name} = \texttt{value}
    ```
  - Self really refers to the object that a method is called on

```python
def main():
    count1 = Counter()
    count2 = Counter()
    x = count1.next_value()
    y = count2.next_value()
```

- `count1` → `self.ticket_num` = 0
- `count2` → `self.ticket_num` = 0
Instance Variables

- Instance variables are variable associated with objects
  - Each object gets its own set of instance variables
  - Generally, they are initialized in constructor for class
  - Instance variables accessed using `self`:
    ```python
    self.variable_name = value
    ```
  - Self really refers to the object that a method is called on

```python
def main():
    count1 = Counter()
    count2 = Counter()
    x = count1.next_value()
    y = count2.next_value()

def next_value(self):
    self.ticket_num += 1
    return self.ticket_num
```

```
count1: self.ticket_num = 0

count2: self.ticket_num = 0
```
• Instance variables are variable associated with objects
  – Each object get its own set of instance variables
  – Generally, they are initialized in constructor for class
  – Instance variables accessed using self:
    
    ```python
    self.\texttt{variable\_name} = \texttt{value}
    ```
  – Self really refers to the object that a method is called on

```python
def main():
    count1 = Counter()
    count2 = Counter()
    x = count1.next_value()
    y = count2.next_value()

def next_value(self):
    self.ticket_num += 1
    return self.ticket_num
```
Instance Variables

- Instance variables are variable associated with objects
  - Each object get its **own set** of instance variables
  - Generally, they are initialized in constructor for class
  - Instance variables accessed using `self`:
    ```python
    self.variable_name = value
    ```
  - Self really refers to the object that a method is called on

```python
def main():
    count1 = Counter()
    count2 = Counter()
    x = count1.next_value()
    y = count2.next_value()
```

```
count1 ➞
    self.ticket_num  1

count2 ➞
    self.ticket_num  0
```
Instance Variables

- Instance variables are variable associated with objects
  - Each object get its **own set** of instance variables
  - Generally, they are initialized in constructor for class
  - Instance variables accessed using `self`:
    ```python
    self.variable_name = value
    ```
  - Self really refers to the object that a method is called on

```python
def main():
    count1 = Counter()
    count2 = Counter()
    x = count1.next_value()
    y = count2.next_value()

def next_value(self):
    self.ticket_num += 1
    return self.ticket_num
```

```plaintext
count1 → self.ticket_num  1
count2 → self.ticket_num  0
```
Instance Variables

- Instance variables are variable associated with objects
  - Each object get its **own set** of instance variables
  - Generally, they are initialized in constructor for class
  - Instance variables accessed using `self`:
    
    ```
    self. variable_name = value
    ```
  - Self really refers to the object that a method is called on

```python
def main():
    count1 = Counter()
    count2 = Counter()
    x = count1.next_value()
    y = count2.next_value()

def next_value(self):
    self.ticket_num += 1
    return self.ticket_num
```

```python
count1 ➔
  self.ticket_num  1

count2 ➔
  self.ticket_num  1
```
Instance Variables

- Instance variables are variable associated with objects
  - Each object gets its own set of instance variables
  - Generally, they are initialized in the constructor for the class
  - Instance variables accessed using `self`:
    ```python
    self.variable_name = value
    ```
  - `Self` really refers to the object that a method is called on

```python
def main():
    count1 = Counter()
    count2 = Counter()
    x = count1.next_value()
    y = count2.next_value()
```

<table>
<thead>
<tr>
<th>count1</th>
<th>self.ticket_num</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>count2</td>
<td>self.ticket_num</td>
<td>1</td>
</tr>
</tbody>
</table>
Methods (Functions) in Class

• Methods (name used for functions in objects)
  – Syntax:
    ```
    def method_name(self, additional parameters):
        body
    ```
  – Works like a regular function in Python
    – Can return values (like a regular function)
    – Has access to `instance` variables (through `self`):
      ```
      self.variable_name = value
      ```
    – Called using an object:
      ```
      object_name.method_name(additional parameters)
      ```
  – Recall, parameter `self` is automatically set by Python as the object that this method is being called on
    • You write: `number = count1.next_value()`
    • Python treats it as: `number = next_value(count1)`
Another Example: Students

• Want a Class to keep track information for Students
  – Each student has information:
    • Name
    • ID number
    • Units completed
  – Want to specify a name and ID number when creating a student object
    • Initially, units completed set to 0
  – Student's number of units completed can be updated over time
  – Also want to be able to check if a student can graduate
    • Student needs to have at least $\text{UNITS_TO_GRADUATE}$ units
Bring Me the Students!
student.py
Learning Goals

1. Learning about Object-Oriented Programming
2. Writing code using Classes and Objects in Python