Intro to Python

Class 4

Review

- Method calls
- Combining lists and dictionaries
- Builtins for collections

Functions on Dictionaries

```
character = {
    'x': 10,
    'y': 20,
    'health': 100,
}

def injure(character, damage):
    character['health'] = character['health'] - damage
    if character['health'] < 0:
        character['health'] = 0

def heal(character, amount):
    character['health'] = character['health'] + amount
    if character['health'] > 100:
        character['health'] = 100
```

Classes

A **class** creates a new type of object.

A class defines the attributes and methods of objects of that type

Classes are used to create new objects of that type

```
class Character():

    def __init__(self, x, y, health):
        self.x = x
        self.y = y
        self.health = health

character = Character(10, 20, 100)
```

A Sense of Self

The first argument to every method is **self**.

self contains the attributes and methods for the current object

```
class Character():

    def __init__(self, x, y, health):
        self.x = x
        self.y = y
        self.health = health

character = Character(10, 20, 100)
```

The __init__ Method

This method defines what the class should do when creating a new object.

```
class Character():

    def __init__(self, x, y, health):
        self.x = x
        self.y = y
        self.health = health

character_a = Character(10, 20, 100)
character_b = Character(10, 20, 100)
```

To create a new Character, the syntax looks like a function call. These arguments are passed to the __init__ method

Class Methods

A class also defines **methods**, which are functions that operate on objects of that type

Assigning values to an attribute on self is how we **mutate** the object's state.

```
# inside the character class

    def heal(self, amount):
        self.health = self.health + amount
        if self.health > 100:
            self.health = 100

    def injure(self, amount):
        self.health = self.health - amount
        if self.health < 0:
           self.health = 0

character = Character(10, 20, 100)
character.injure(10)
```

Let's Develop It

- In your text editor, create your own class with an __init__ method, and at least one other method.
- Open a Python shell and import the class. Create one or more objects from the class
- If time allows, create a function that creates objects from your class, calls a method, and prints one of its attributes
- Use the next slide as an example

```
# in character.py
class Character():

    def __init__(self, x, y, health):
        self.x = x
        self.y = y
        self.health = health

    def heal(self, amount):
        self.health = self.health + amount
        if self.health > 100:
            self.health = 100
```

```
# in Python shell
from character import Character
character_a = Character(10, 20, 100)
character_a.injure(10)
print "character health is: " + character_a.health
```

Inheritance

A class can **inherit** from another class.

A class that inherits from another is called the "child class" and obtains the methods and attributes of its "parent"

```
class Mobile(object):
    """
    An object with an x, y position, and methods for moving
    """

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def move_up():
        self.y = self.y - 1

    # ... methods for move_down, move_left, and move_right
```

Inheritance Continued

The move_up method is **overridden** in the child class below:

```
class BoundedMobile(Mobile):
    """
    An object with an x, y position, and methods for moving
    The x, y position must be within bounds
    """

    def move_up():
        self.y = self.y - 1
        if self.y < 0:
            self.y = 0
```

See **mobile.py** for a more complete example.

What's Super about Super

super is often helpful when writing methods that override the method of the parent class

```
class BoundedMobile(Mobile):
    """
    An object with an x, y position, and methods for moving
    The x, y position must be within bounds
    """

    def move_up():
        super(BoundedMobile, self).move_up()
        if self.y < 0:
            self.y = 0
```

The call to super() takes the name of the child class, followed by self. This is followed by the method call and any arguments to pass to it

Composition

Classes can also use the technique of **composition**

This simply means that a given object contains other objects within it.

This often leads to a clearer and simpler design

```
class Grid(object):

    def __init__(self, x_limit, y_limit):
        self.x_limit = x_limit
        self.y_limit = y_limit
        self.mobiles = []

    def add_mobile(self, x, y):
        mob = BoundedMobile(x, y, self.x_limit, self.y_limit)
        mobs = self.mobiles.get((x, y), [])
        mobs.append(mob)
        self.mobiles[(x, y)] = mobs
```

Composition Continued

Given the class on the previous slide, the following code creates mobiles within the grid object. (Complete code is available in the aforementioned **mobile.py** file.)

```
from mobile import Grid

grid = Grid(7, 7)
grid.add_mobile(1, 2)
grid.add_mobile(0, 1)
grid.add_mobile(0, 1)

grid.display_grid()
```

Let's Develop It

Create a class that uses inheritance, composition, or both.

To help you, use your work from the last exercise or the classes from **mobile.py**

Higher order functions

A **higher order function** is a function that returns a function, takes a function as an argument, or both

One commonly used higher order function that is a Python builtin is called **map**

```
# Define any function
def sqaure(number):
    return number ** 2

# Pass the function to map along with an iterable
squares = map(square, range(10))
```

N.B. - map has performance problems for large data sets and should only be used when the data set is well defined and somewhat small.

Let's Develop It

Choose among any of these projects (Resources available on the next page):

- Search the Web - Write a program that searches the web using DuckDuckGo and displays results.
- Encryption - Write a program that encrypts a string from user input, or file and is able to decrypt it as well.
- Command Line Game - Create a simple game that runs inside the terminal.

Let's Develop It Resources

Search the Web	**python-duckduckgo** library to get started. Download duckduckgo.py and put it in the same directory as your code. Use the query() function it provides to begin. (HINT: Results are often empty, but 'related' list usually has a few hits.)
Encryption	Read about the **Caesar Cipher** or find a similarly simple encryption mechanism online. You should find

the ord() and chr() functions helpful, as well as the modulus operator '%'

continued on next page...

Let's Develop It Resources Continued

Command Line Game	This might be a text adventure with paragraphs of text followed by a series of choices for the user. A choice maps to another node in the story (another paragraph with choices). You might try storing the paragraphs separately in a text file. The format might be something different, such as a series of "rooms", each with a description, for

the user to explore by entering commands such as "go west". Examples of these kinds of games are **Colossal Cave Adventure** and **Zork**

Future Resources

Python.org Documentation	Official Python Documentation
Think Python	Online and print book with exercises.
Learn Python the Hard Way	Online and print book with exercises
Google's Python Class	Video lectures coupled with exercises
New Coder	Ideas for slightly larger projects and resources to get you started. Projects include accessing API's,

	scraping pages, writing IRC bots, and others.
Girl Develop It	Local workshops, events, and coding sessions

Questions?