Week 8

Classes and Objects

Special thanks to Scott Shawcroft, Ryan Tucker, and Paul Beck for their work on these slides.

OOP, Defining a Class

- Python was built as a procedural language
  - OOP exists and works fine, but feels a bit more "tacked on"
  - Java probably does classes better than Python (gasp)

- Declaring a class:

```
class name:
    statements
```

Fields

`name = value`

- Example:

```
class Point:
    x = 0
    y = 0

# main
p1 = Point()
p1.x = 2
p1.y = -5
```

point.py

```
class Point:
    x = 0
    y = 0
```

- can be declared directly inside class or in constructors

- Python does not really have encapsulation or private fields
  - relies on caller to "be nice" and not mess with objects' contents

Using a Class

`import` **class**

- client programs must import the classes they use

point_main.py

```python
from Point import *

# main
p1 = Point()
p1.x = 7
p1.y = -3

...
```

"Implicit" Parameter (self)

- Java: `this`, implicit

```
public void translate(int dx, int dy) {
    x += dx;      // this.x += dx;
    y += dy;      // this.y += dy;
}
```

- Python: `self`, explicit
  - `self` must be the first parameter to any object method
  - *must* access the object's fields through the `self` reference

```
def translate(self, dx, dy):
    self.x += dx
    self.y += dy
...
```

Methods

```
def name(self, parameter, ..., parameter):
    statements
```

- additional **parameters** are optional

- Example:

```
class Point:
    def translate(self, dx, dy):
        self.x += dx
        self.y += dy
    ...
```

- Exercise: Write `distance` and `distance_from_origin` .

Exercise Answer

point.py

```
from math import *

class Point:
    x = 0
    y = 0

    def distance_from_origin(self):
        return sqrt(self.x * self.x + self.y * self.y)

    def distance(self, other):
        dx = self.x - other.x
        dy = self.y - other.y
        return sqrt(dx * dx + dy * dy)
```

Constructors

```
def __init__(self, parameter, ..., parameter):
    statements
```

- a constructor is a special method with the name `__init__`
- Example:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    ...
```

toString and __str__

```
def __str__(self):
    return string
```

- equivalent to Java's `toString` (converts object to a string)
- invoked automatically when `str` or `print` is called

```
def __str__(self):
    return "(" + str(self.x) + ", " + str(self.y) + ")"
```

- Others: define a < on your class by writing `__lt__` , etc.:
  http://docs.python.org/ref/customization.html

Complete Point Class

point.py

```python
from math import *

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

    def distance_from_origin(self):
        return sqrt(self.x * self.x + self.y * self.y)

    def distance(self, other):
        dx = self.x - other.x
        dy = self.y - other.y
        return sqrt(dx * dx + dy * dy)

    def translate(self, dx, dy):
        self.x += dx
        self.y += dy

    def __str__(self):
        return "(" + str(self.x) + ", " + str(self.y) + ")"
```

Inheritance

```
class name(superclass):
    statements
```

- Example:

```
class Point3D(Point):    # Point3D extends Point
    z = 0
    ...
```

- Python also supports *multiple inheritance*

```
class name(superclass, ..., superclass):
    statements
```

Calling Superclass Methods

- methods: **class**.**method**(**parameters**)
- constructors: **class**.__init__(**parameters**)

```
class Point3D(Point):
    z = 0
    def __init__(self, x, y, z):
        Point.__init__(self, x, y)
        self.z = z

    def translate(self, dx, dy, dz):
        Point.translate(self, dx, dy)
        self.z += dz
```

The pyGame Package

- A set of Python modules to help write games
- Deals with media (pictures, sound) nicely
- Interacts with user nicely (keyboard, joystick, mouse input)

Where to Start?

- The official pyGame website
- Search for tutorials
- The Application Programming Interface (API)
  - specifies the classes and functions in package
- Experiment!

A Skeleton

- Tutorials basically all have the same setup -- let's use it!

template.py

```
from pygame import *
from pygame.sprite import *
from random import *

init()

screen = display.set_mode((640, 480))
display.set_caption('Whack-a-mole')

while True:
    e = event.poll()
    if e.type == QUIT:
        quit()
        break

    screen.fill(Color("white"))
    display.update()
```

Surface

- All images are represented as `Surfaces`
- `display.set_mode(x, y)` returns a Surface object
- `fill("`**color**`")` fills the object it's called on
- `blit(`**surface**`, `**area**`)` paints **surface** onto the object it's called on in the rectangle bounded by **area**

Rect

- Objects that store rectangular coordinates
- `center` holds the object's center as a tuple
- `colliderect`**(target)** returns True if the parameter overlaps with the object
- `collidepoint(`**target**) returns True if the target point overlaps with the object

Media

- Loading an image:
  - `img = image.load("file.gif").convert()`
- Getting a bounding rectangle:
  - `img_rect = img.get_rect()`
- Loading and playing a sound file:
  - `mixer.Sound("file.wav").play()`

Sprite

- Class visible game objects inherit from

Ball.py

```python
from pygame import *
from pygame.sprite import *

class Ball(Sprite):
    def __init__(self):
        Sprite.__init__(self)
        self.image = image.load("ball.png").convert()
        self.rect = self.image.get_rect()

    def update(self):
        self.rect.center = mouse.get_pos()
```

Using Sprites

- They're just objects: initialize them
  - `ball = Ball()`

- Create a group of sprites in main
  - `sprites = RenderPlain(`**sprite1**`, `**sprite2**`)`

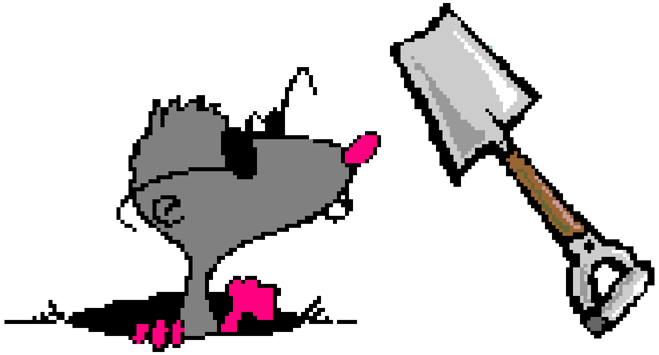
- Groups know how to draw and update
  - `sprites.update()`
  - `sprites.draw(`**surface**`)`

Exercise: Whack-a-mole

- Clicking on the mole
  - plays a sound
  - makes the mole move

- The number of hits is displayed at the top of the screen

- For version 2, hit the mole with a shovel



Using Resources

- You should now be more comfortable with using APIs
- Never be afraid to experiment!
- The Python community is very open to questions.

SciPy

- Math, science, engineering tools
- Official website (http://www.scipy.org/)
- Installation (http://www.scipy.org/Installing_SciPy)
- Cookbook (http://www.scipy.org/Cookbook)
- Tutorial (http://www.tau.ac.il/~kineret/amit/scipy_tutorial/)
- API (http://www.scipy.org/doc/api_docs/)

python™

Django

- Web application framework
- Official website (http://www.djangoproject.com/)
- Free book (http://www.djangobook.com/)
- API (http://www.djangoproject.com/documentation/db-api/)

So Many Packages!

- Official listing (http://pypi.python.org/pypi?%3Aaction=browse)
- If it doesn't exist, make your own!

The sky's the limit!