Classes & Objects

A class combines (and abstracts) data and functions

An object is an instantiation of a class
Classes & Objects

class

object
A class combines (and abstracts) data and functions

An object is an instantiation of a class

*String* is a built-in class, *append* is a function

*Int* is a built-in class, *+* is a function

We can define our own classes
myball = Ball(10.0, 15.0, 0.0, -5.0)

constructor:
Classes & Objects

myball = Ball(10.0, 15.0, 0.0, -5.0)

constructor:
• allocate memory for a Ball object
Classes & Objects

myball = Ball(10.0, 15.0, 0.0, -5.0)

constructor:
- allocate memory for a Ball object
- initializes the Ball object with values
Classes & Objects

myball = Ball(10.0, 15.0, 0.0, -5.0)

class constructor:
- allocate memory for a Ball object
- initializes the Ball object with values
- returns address of the Ball object
myball = Ball(10.0, 15.0, 0.0, -5.0)

constructor:
  • allocate memory for a Ball object
  • initializes the Ball object with values
  • returns address of the Ball object
  • similar to a list
Classes & Objects

myball = Ball(10.0, 15.0, 0.0, -5.0)
Classes & Objects

ball1 = Ball(10.0, 15.0, 0.0, -5.0)
ball2 = Ball(12.0, 23.0, 2.0, 3.0)
ball1 = Ball(10.0, 15.0, 0.0, -5.0)
bball2 = Ball(12.0, 23.0, 2.0, 3.0)

print( "the x-coordinate is ", ball1.x)
ball1 = Ball(5.0, 4.0, 3.0, 6.0)
print( "the x-coordinate is ", ball1.x)  #5.0

ball1.update_position(0.1)
print( "the x-coordinate is ", ball1.x)  #5.3
import drawSvg as draw

D = draw.Drawing(200, 200, origin='center') # define drawing canvas

EARTH_GRAVITY_ACCELERATION = -9.8  # acceleration due to gravity, m/sec^2
BALL_RADIUS = 10  # radius of the ball in pixels

class Ball:
    def __init__(self, start_x, start_y, start_v_x, start_v_y, color='blue'):
        # Ball location and velocity
        self.x = start_x
        self.y = start_y
        self.v_x = start_v_x
        self.v_y = start_v_y

        # Ball color, for drawing purposes
        self.color = color

    def update_position(self, timestep):
        self.x = self.x + timestep * self.v_x
        self.y = self.y + timestep * self.v_y

    def update_velocity(self, timestep):
        self.v_y = self.v_y + timestep * EARTH_GRAVITY_ACCELERATION

    def animate_step(self, timestep):
        self.update_position(timestep)
        self.update_velocity(timestep)

    def draw(self):
        D.append(draw.Circle(self.x, self.y, BALL_RADIUS, fill=self.color))
import drawSvg as draw

D = draw.Drawing(200, 200, origin='center')  # define drawing canvas
EARTH GRAVITY ACCELERATION = -9.8  # acceleration due to gravity, m/sec^2
BALL_RADIUS = 10  # radius of the ball in pixels

class Ball:
    def __init__(self, start_x, start_y, start_v_x, start_v_y, color='blue'):
        # Ball location and velocity
        self.x = start_x
        self.y = start_y
        self.v_x = start_v_x
        self.v_y = start_v_y

        # Ball color, for drawing purposes
        self.color = color

    def update_position(self, timestep):
        self.x = self.x + timestep * self.v_x
        self.y = self.y + timestep * self.v_y

    def update_velocity(self, timestep):
        self.v_y = self.v_y + timestep * EARTH GRAVITY ACCELERATION

    def animate_step(self, timestep):
        self.update_position(timestep)
        self.update_velocity(timestep)

    def draw(self):
        D.append(draw.Circle(self.x, self.y, BALL_RADIUS, fill=self.color))
import drawSvg as draw

D = draw.Drawing(200, 200, origin='center')  # define drawing canvas
EARTH_GRAVITY_ACCELERATION = -9.8  # acceleration due to gravity, m/sec^2
BALL_RADIUS = 10  # radius of the ball in pixels

class Ball:
    def __init__(self, start_x, start_y, start_v_x, start_v_y, color='blue'):
        # Ball location and velocity
        self.x = start_x
        self.y = start_y
        self.v_x = start_v_x
        self.v_y = start_v_y

        # Ball color, for drawing purposes
        self.color = color

    def update_position(self, timestep):
        self.x = self.x + timestep * self.v_x
        self.y = self.y + timestep * self.v_y

    def update_velocity(self, timestep):
        self.v_y = self.v_y + timestep * EARTH_GRAVITY_ACCELERATION

    def animate_step(self, timestep):
        self.update_position(timestep)
        self.update_velocity(timestep)

    def draw(self):
        D.append(draw.Circle(self.x, self.y, BALL_RADIUS, fill=self.color))
import drawSvg as draw

D = draw.Drawing(200, 200, origin='center') # define drawing canvas
EARTH_GRAVITY_ACCELERATION = -9.8 # acceleration due to gravity, m/sec^2
BALL_RADIUS = 10 # radius of the ball in pixels

class Ball:
    def __init__(self, start_x, start_y, start_v_x, start_v_y, color='blue'):
        # Ball location and velocity
        self.x = start_x
        self.y = start_y
        self.v_x = start_v_x
        self.v_y = start_v_y

        # Ball color, for drawing purposes
        self.color = color

    def update_position(self, timestep):
        self.x = self.x + timestep * self.v_x
        self.y = self.y + timestep * self.v_y

    def update_velocity(self, timestep):
        self.v_y = self.v_y + timestep * EARTH_GRAVITY_ACCELERATION

    def animate_step(self, timestep):
        self.update_position(timestep)
        self.update_velocity(timestep)

    def draw(self):
        D.append(draw.Circle(self.x, self.y, BALL_RADIUS, fill=self.color))
myball = Ball(10.0, 15.0, 0.0, -5.0)

def __init__(self, start_x, start_y, start_v_x, start_v_y, color='blue'):
    # Ball location and velocity
    self.x = start_x
    self.y = start_y
    self.v_x = start_v_x
    self.v_y = start_v_y

    # Ball color, for drawing purposes
    self.color = color
myball = Ball(10.0, 15.0, 0.0, -5.0)

myball.update_position(0.1)  # update_position(myball, 0.1)

def update_position(self, timestep):
    self.x = self.x + timestep * self.v_x  # myball.x = myball.x + ...
    self.y = self.y + timestep * self.v_y
[ lecture07-1.ipynb ]
## Lists of Objects

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ball_list</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x: 10.0</td>
<td>x: 20.0</td>
<td>x: 5.0</td>
</tr>
<tr>
<td>y: 15.0</td>
<td>y: 35.0</td>
<td>y: 5.0</td>
</tr>
<tr>
<td>v_x: 0.0</td>
<td>v_x: 0.0</td>
<td>v_x: 1.0</td>
</tr>
<tr>
<td>v_y: -5.0</td>
<td>v_y: 0.0</td>
<td>v_y: 2.0</td>
</tr>
<tr>
<td>color:'r'</td>
<td>color:'g'</td>
<td>color:'b'</td>
</tr>
</tbody>
</table>
### Lists of Objects

**ball_list**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>x:</td>
<td>10.0</td>
<td>20.0</td>
</tr>
</tbody>
</table>
y:| 15.0 | 35.0 | 5.0 |
v_x:| 0.0 | 0.0 | 1.0 |
v_y:| -5.0 | 0.0 | 2.0 |
color: | 'r' | 'g' | 'b' |

```python
b = ball_list[0]
b.y
```
Lists of Objects

ball_list

ball_list[0].y
# --- DRILL ----
# adapt the appropriate Ball member function(s) so
# that each ball bounces off the “floor”
# --- DRILL ----
# adapt the appropriate Ball member function(s) so
# that each ball bounces off the “floor”

[ lecture07-1.ipynb ]
b = Ball(1,2,3,4,0,0,1)
print b

<ball.Ball instance at 0x100499a70>
def __str__(self):
    return str(self.x) + ', ' + str(self.y)

b = Ball(0,0,1,-1)
print(b)
1, 2
Write a class BankAccount. This class has one instance variable "balance" and four methods each of which requires only one line of code each:

- **constructor**: take as input an amount and initialize "balance"
- **deposit**: take as input an amount and increment "balance"
- **withdraw**: take as input an amount and decrement "balance"
- **overdrawn**: return True if "balance" is less than 0, else return False

Write a small driver code to test all functions
Write a class BankAccount. This class has one instance variable "balance" and four methods each of which requires only one line of code each:

- **constructor**: take as input an amount and initialize "balance"
- **deposit**: take as input an amount and increment "balance"
- **withdraw**: take as input an amount and decrement "balance"
- **overdrawn**: return True if "balance" is less than 0, else return False

Write a small driver code to test all functions

[ lecture07-1.ipynb ]
Write a class named Kangaroo with:

- **constructor**: initializes an instance variable named "pouch_contents" to an empty list.
- **put_in_pouch**: takes a string as input and adds it to pouch_contents, if it is not already in the pouch. If it is already in the pouch, print "object already in pouch"
- **__str__**: prints content of pouch. If the pouch is empty then print "The kangaroo's pouch is empty." If the pouch is not empty then print "The kangaroo’s pouch contains: [A,B,C]" (where A, B, C are the contents of the pouch).

Write a short driver that tests your class
# --- DRILL ----
# Write a class named Kangaroo with:
#
# constructor: initializes an instance variable named "pouch_contents" to an empty list.
#
# put_in_pouch: takes a string as input and adds it to pouch_contents, if it is not already in the pouch. If it is already in the pouch, print "object already in pouch"
#
# __str__: prints content of pouch. If the pouch is empty then print "The kangaroo's pouch is empty." If the pouch is not empty then print "The kangaroo’s pouch contains: [A,B,C]" (where A, B, C are the contents of the pouch).

[ lecture07-1.ipynb ]