Supplemental Lecture 5

Raphael Baer-Way

Object Oriented Programming

Object Oriented Programming

- Arguably one of the most **powerful** features of python
- Allows us to construct **objects** with their own **traits** and **methods**
- We can create a **Class** of objects, once you make an object that object will have all the traits and methods of its' class.

Example

Class Human

Traits: ← Properties of the object

- eye color, hair color, skin color, height, weight, etc….

Example

Class Human

Traits: ← Properties of the object

- eye color, hair color, skin color, height, weight, etc….

Methods: ← Things the object can do

- Jump, run, clap, skip, push, talk, think, etc….

Example

Define the class

```python
class Human:
    def __init__(self, age, height):
        self.age = age
        self.height = height
    def grow(self):
        self.height += 1
```

Example

Define the class

“Constructor” function

```
class Human:
    def __init__(self, age, height):
        self.age = age
        self.height = height
    def grow(self):
        self.height += 1
```

Example

Define the class

“Constructor” function

Traits

```python
class Human:
    def __init__(self, age, height):
        self.age = age
        self.height = height
    def grow(self):
        self.height += 1
```

Example

```
class Human:

    def __init__(self, age, height):

        self.age = age

        self.height = height

    def grow(self):

        self.height += 1
```

Define the class

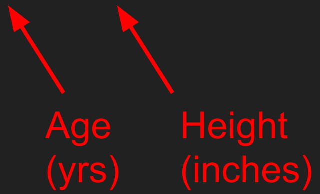
“Constructor” function

Traits

Method for an object in the class

Example

```
>>> James = Human(20, 70)
```



Age (yrs)

Height (inches)

The line above creates an object from the human class with the required arguments of the constructor function. I call this object: James

Example

```
>>> James = Human(20, 70)
```

Age (yrs)

Height (inches)

The line above creates an object from the human class with the required arguments of the constructor function. I call this object: James

```
>>>James.grow()
```

New notation for calling a function!
Does it look familiar?

The line above calls the grow method belonging to an object in the human class, which just adds an inch to my height so now

```
>>> print(James.height)
```

```
71
```

You’ve already used object oriented programming!

Anyone remember this?

```
ax.plot(x,y)

ax.set_title(‘Title’)

ax.set_xlabel(‘x-axis’)

ax.set_ylabel(‘y-axis’)

ax.legend()
```

These are all just **methods** for the ax object!

You’ve already used object oriented programming!

Anyone remember this?

```
ax.plot(x,y)

ax.set_title(‘Title’)

ax.set_xlabel(‘x-axis’)

ax.set_ylabel(‘y-axis’)

ax.legend()
```

Even with these

```
list.append()
list.remove()
  np.array()
```

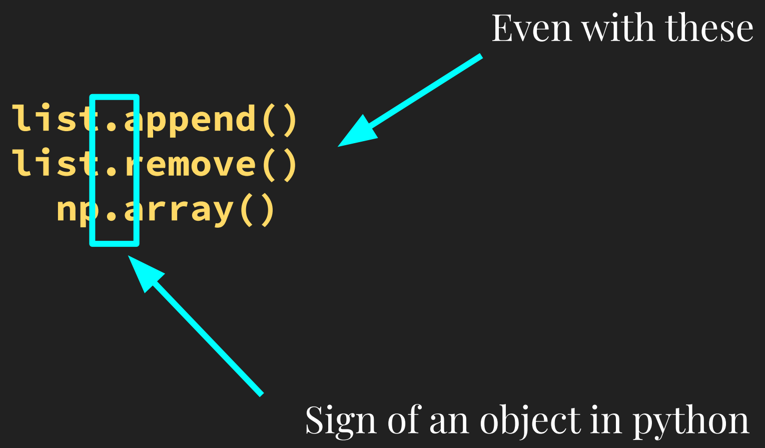
These are all just **methods** for the ax object!

You’ve already used object oriented programming!

Anyone remember this?

```
ax.plot(x,y)

ax.set_title(‘Title’)

ax.set_xlabel(‘x-axis’)

ax.set_ylabel(‘y-axis’)

ax.legend()
```

These are all just **methods** for the ax object!



Even with these

```
list.append()
list.remove()
  np.array()
```

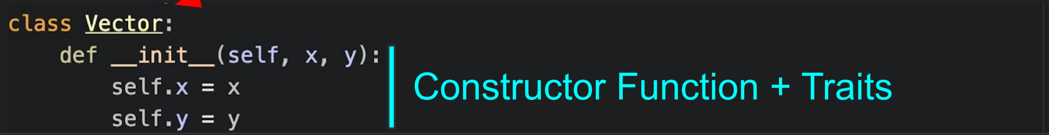
Sign of an object in python

More Objects you’ve used!

- Dataframe from Pandas
- Numpy arrays
- Dictionaries
- Axes in matplotlib
- Lists and Tuples
- Can you think of anything else?

Can be useful for physics!

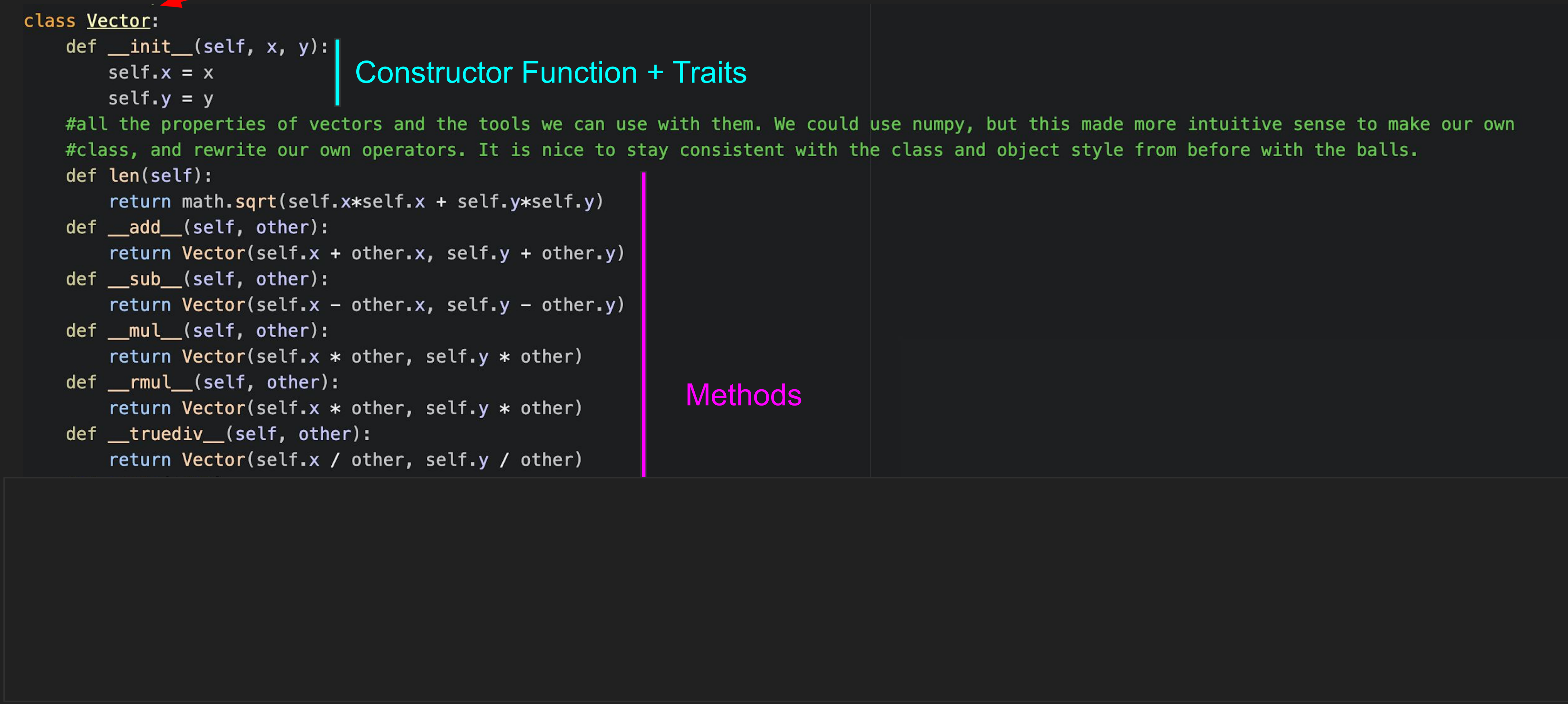
Vector Class



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

Constructor Function + Traits

Vector Class

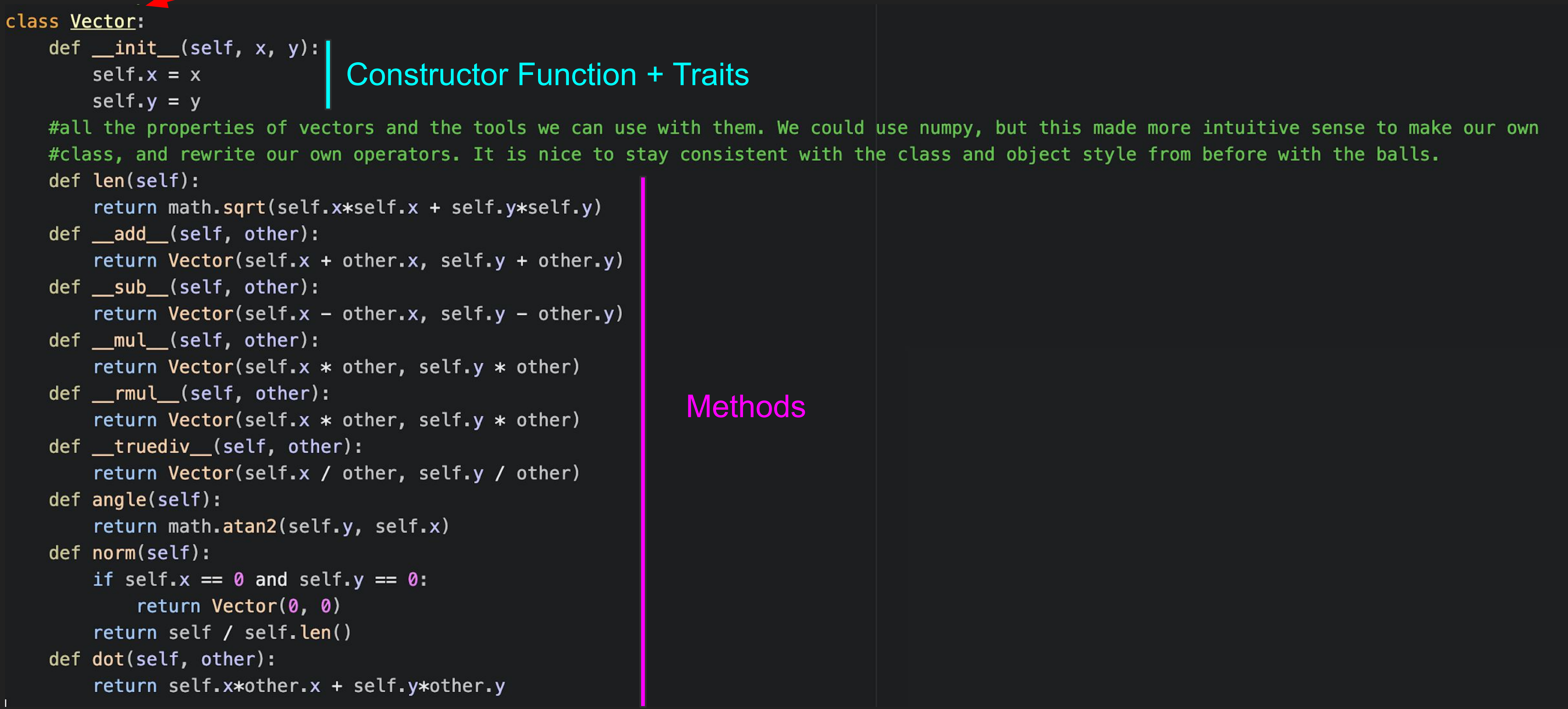


```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    #all the properties of vectors and the tools we can use with them. We could use numpy, but this made more intuitive sense to make our own
    #class, and rewrite our own operators. It is nice to stay consistent with the class and object style from before with the balls.
    def len(self):
        return math.sqrt(self.x*self.x + self.y*self.y)
    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)
    def __sub__(self, other):
        return Vector(self.x - other.x, self.y - other.y)
    def __mul__(self, other):
        return Vector(self.x * other, self.y * other)
    def __rmul__(self, other):
        return Vector(self.x * other, self.y * other)
    def __truediv__(self, other):
        return Vector(self.x / other, self.y / other)
```

Constructor Function + Traits

Methods

Vector Class



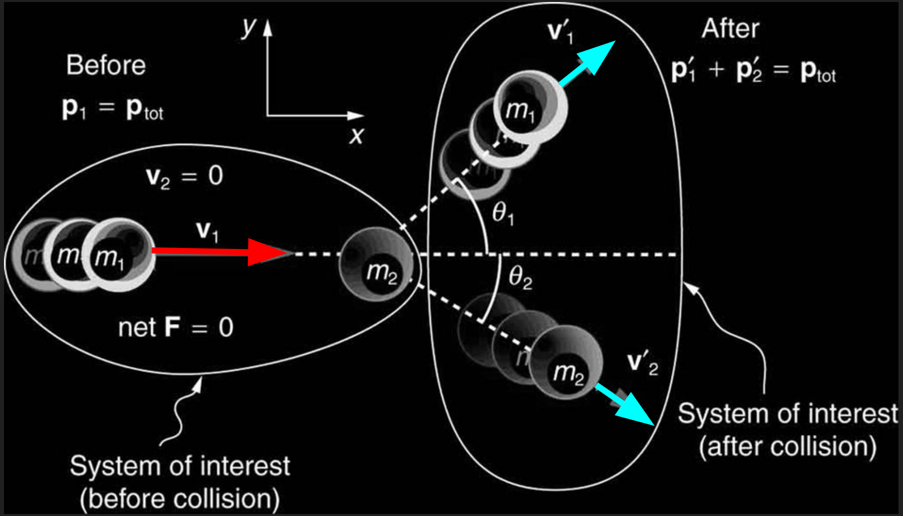
```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    #all the properties of vectors and the tools we can use with them. We could use numpy, but this made more intuitive sense to make our own
    #class, and rewrite our own operators. It is nice to stay consistent with the class and object style from before with the balls.
    def len(self):
        return math.sqrt(self.x*self.x + self.y*self.y)
    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)
    def __sub__(self, other):
        return Vector(self.x - other.x, self.y - other.y)
    def __mul__(self, other):
        return Vector(self.x * other, self.y * other)
    def __rmul__(self, other):
        return Vector(self.x * other, self.y * other)
    def __truediv__(self, other):
        return Vector(self.x / other, self.y / other)
    def angle(self):
        return math.atan2(self.y, self.x)
    def norm(self):
        if self.x == 0 and self.y == 0:
            return Vector(0, 0)
        return self / self.len()
    def dot(self, other):
        return self.x*other.x + self.y*other.y
```

Constructor Function + Traits

Methods

Cool Example

You can use that vector class when doing physics calculations!



https://cnx.org/resources/210cd3c178096cca924cd3577ccf01460a1c752b

Demo-

https://colab.research.google.com/drive/1HM9f0b70YWBSv8859sYpcQ4AnWThJC3x