
Python DeCal
Week 8

Announcements

- Final Projects
  - Communicating with your group members
  - Final Project Proposal due: **Wednesday**
    - **Submit a PDF**
    - **You don’t have to stick with it but it’s recommended that you do**
- Attendance: https://forms.gle/d5Bu8TdshuvuSxtN7

PANDAS

- One of the easiest ways to import data files

pandas

Uses a new object called a dataframe

Importing Pandas

```
import pandas as pd
```

- This library imports data files with tremendous ease by storing them in a new data type called a **dataframe**

	count	names	percentage
0	19837	Liam	0.0102
1	18688	Emma	0.0101
2	18267	Noah	0.0097
3	17921	Olivia	0.0095
4	14924	Ava	0.0081

Uses: Reading in Data

- Reading in data from a csv file

```
Data_frame = pd.read_csv('filename.csv')
```

- Reading in data from a excel file

```
Data_frame = pd.read_excel('filename.excel')
```

- Reading in data from a hdf5 file

```
Data_frame = pd.read_hdf('filename.hdf')
```

Uses: DataFrames to Arrays/Lists

- Now how do we extract a column from our data?

```
Column_values = Data_frame['column name']
```

- If you want to make the data values a numpy array:

```
Column_values = Data_frame['column name'].to_numpy()
```

- If you want to make the data values a regular list:

```
Column_values = Data_frame['column name'].to_list()
```

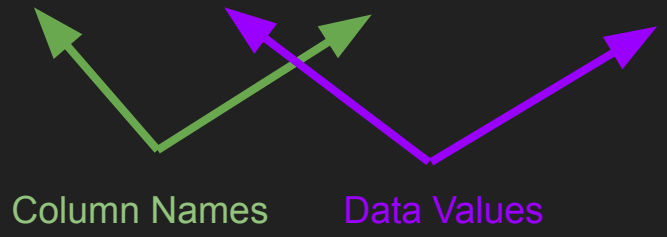
Making Your Own DataFrames

- You have some data lists or arrays:

```
names = ['Jim', 'Dwight', 'Pam', 'Michael', 'Stanley']

birth_count = [9837, 8688, 8267, 1921, 1924]
```

- Lets first construct a dictionary:

```
data = {'names': names, 'count': birth_count}
```



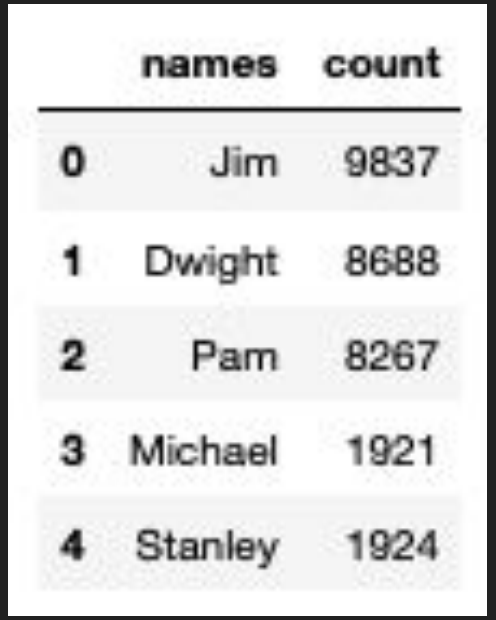
Data Conversion

- Now we can convert this data to a DataFrame:

```
>>> df = pd.DataFrame(data)
```

- Lastly we can export our DataFrame to a .csv file:

```
>>> df.to_csv('filename.csv')
```



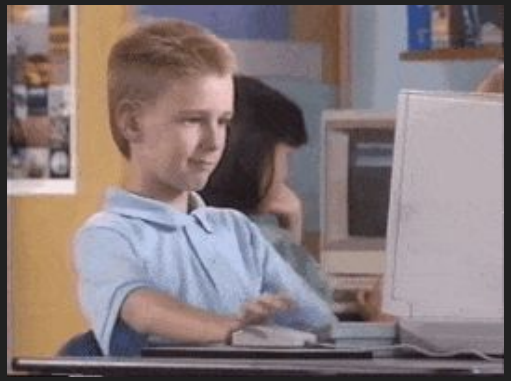
	names	count
0	Jim	9837
1	Dwight	8688
2	Pam	8267
3	Michael	1921
4	Stanley	1924

You're Done!

That's the basics, we can import and export data with only a handful of functions.

If you ever want to see if pandas can do something, google it! This library is **Extremely** well documented

CLICK HERE FOR FULL DOCUMENTATION GUIDE



Final Project Proposal Help

- With the remaining time we are going to put everyone in breakout rooms and help make sure you have a plan together and your proposals are coming together for wednesday
- If you’re done it will be open office hours for HW 6 due Friday

Wednesday

https://forms.gle/RxF66Y5nMwemSswh7

Announcements

- We gave y’all some more time for final project proposals
- Last Python HW is due this Friday
- Project Proposals are due on Wednesday after Break

Breakout Rooms

- What is a DataFrame object?
- Are DataFrames standard python objects?

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

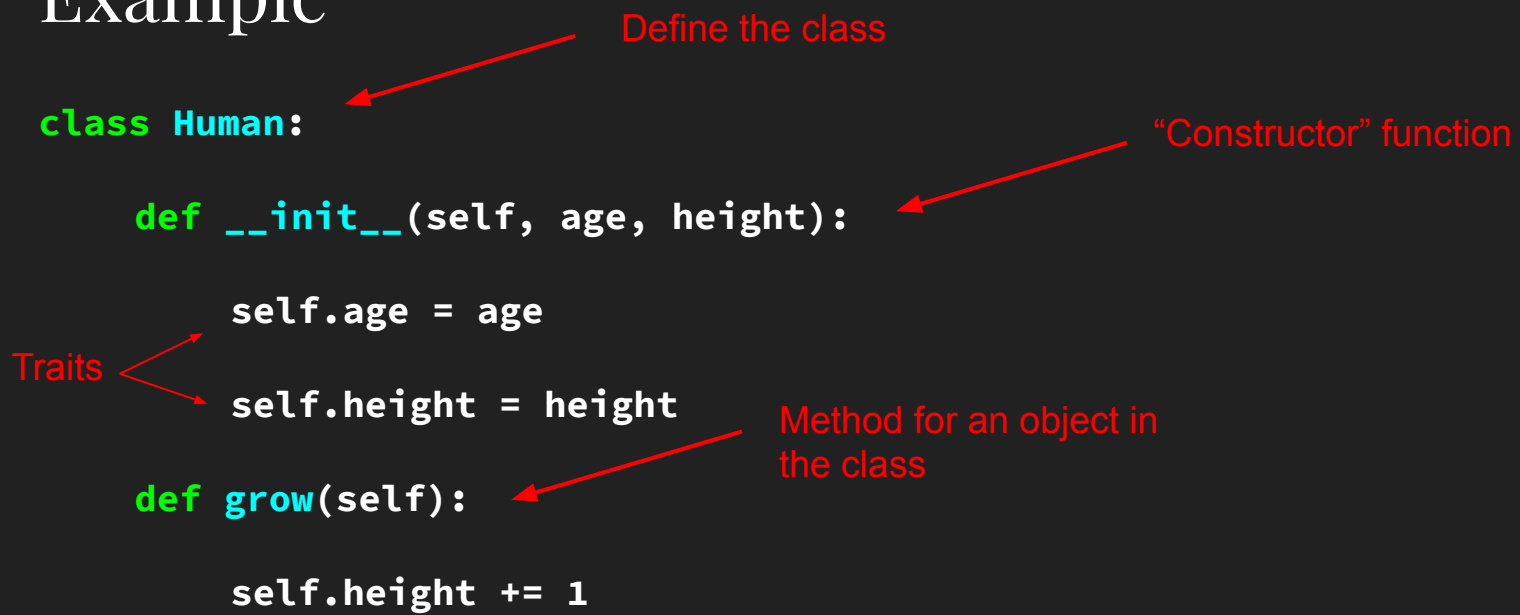
Traits:  Properties of the object

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

Methods:  Things the object can do

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

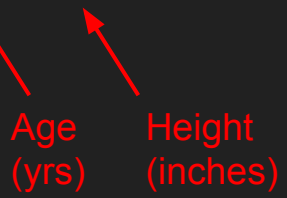
Example




```
class Human:

    def __init__(self, age, height):

        self.age = age

        self.height = height

    def grow(self):

        self.height += 1
```

Example

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



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

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

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

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()
```

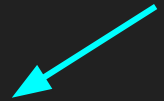
These are all just **methods** for the ax object!

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



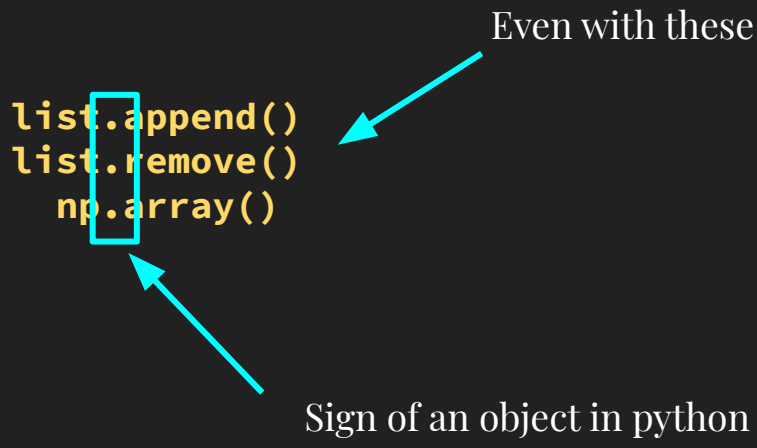
```
list.append()
list.remove()
  np.array()
```

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

You've already used object oriented programming!

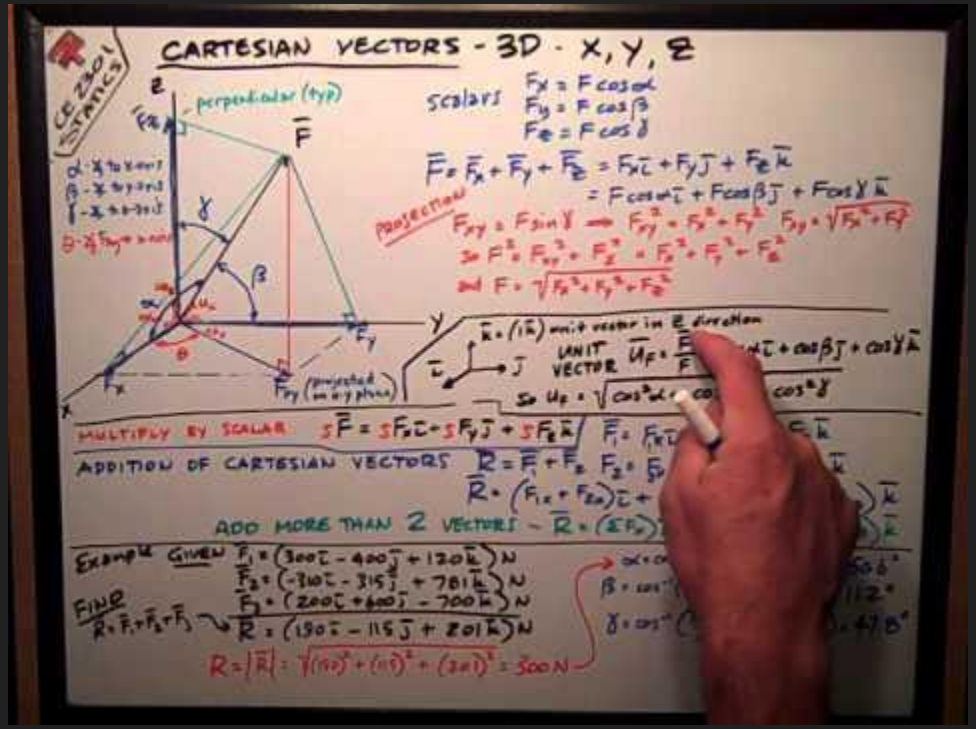
Anyone remember this?

```
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!

Can be useful for physics!



Can be useful for physics!

Vector Class

```
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

Demo