GENERAL ASSEMBLY

Object-Oriented Programming/Intro to Python

Joseph Nelson, h/t Haley Boyan
Data Science Immersive, GA DC

AGENDA

- Introduction/Confirm Install
- What is Python?
- Basic Python Data Types
- Python Classes

Learning Objectives

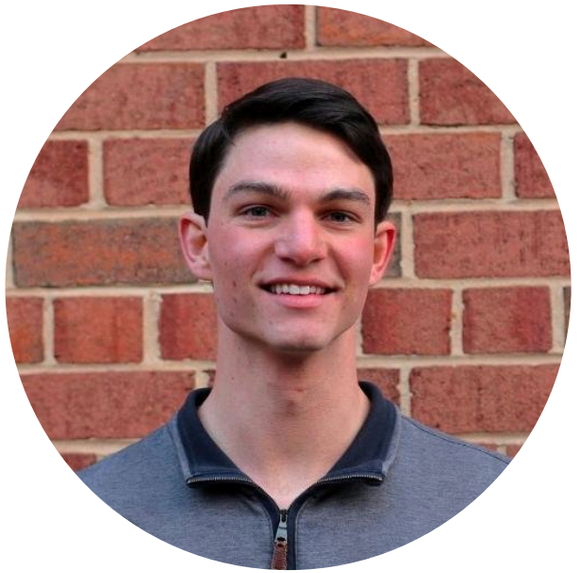
- Describe, qualitatively, the value of knowing Python and why its usage has been core to data analysis and data science
- Know and use the different variable types in Python
- Define what object-oriented programming is, its advantages, and use them in Python
- Gain exposure to Python development tools

“All models are wrong. Some are useful.”

— George Box, 1978

ABOUT

- Data Science Immersive Instructor
- **From:** Des Moines, Iowa
- **Influences:** Marc Andreessen & Ben Horowitz, Zuckerberg, Andrew Ng, Yann LeCun, Jürgen Schmidhuber
- **Likes:** Hockey, SaaS, bad data science puns, running



You

- Python/coding exposure (0 to 5): some Codecademy, workshops, built anything?
- Why you need to learn Python (briefly)
- *Our TA will circulate to help you confirm your installations at this point

Why Python?

What is Python?

- When I say Python, what comes to mind?

- *Our TA will circulate to help you confirm your installations at this point

What is Python?

- Python is a **high-level**, **object-oriented**, **open source**, **software development language** also commonly used (and developed for) **scientific computing**.

- *Our TA will circulate to help you confirm your installations at this point

Python Data Types

Instead of just talking about variables…

- We'll do a walkthrough of different variables in a Jupyter Notebook

INTRODUCTION: CLASSES

OBJECT ORIENTED PROGRAMMING

- What are some objects that we have used in Python?
- What makes up an object?
- What is the difference between a function and a variable?

WHY USE CLASSES

- Have a way to create many variations of a single object
- A simple way to avoid redundancy in code
- Avoid completely recreating something that already exists if you want to add features
- Group related objects together

WHAT IS A CLASS

- Think of a class as a blueprint
- It isn't something in itself, it simply describes how to make something.
- You can create lots of objects from that blueprint
    - These are called **instances**

- Classes have:
    - **Attributes** (Descriptions, variables)
    - **Methods** (Functions, things the object can do)

CREATING A CLASS

- Create a class using the “class” operator:

```
class class_name:
    [statement 1]
    [statement 2]
    [statement 3]
    [etc]
```

- What you have created is a description of an object (the variables) and what operations you can do with the shape (the functions)
- You have **not** made an actual object, just the description of what that object is.
- No code is run when you define a class

CLASS EXAMPLE

- Look at series.py
- This is the python code defining a pandas series
- A series is just a class - made up of attributes and methods

SIMPLER CLASS EXAMPLE

```
class Shape:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    description = "This shape has not been described yet"
    author = "Nobody has claimed to make this shape yet"
    def area(self):
        return self.x * self.y
    def perimeter(self):
        return 2 * self.x + 2 * self.y
    def describe(self,text):
        self.description = text
    def authorName(self,text):
        self.author = text
    def scaleSize(self,scale):
       self.x = self.x * scale
       self.y = self.y * scale
```

SIMPLER CLASS EXAMPLE

```
class Shape:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    description = "This shape has not been described yet"
    author = "Nobody has claimed to make this shape yet"
    def area(self):
        return self.x * self.y
    def perimeter(self):
        return 2 * self.x + 2 * self.y
    def describe(self,text):
        self.description = text
    def authorName(self,text):
        self.author = text
    def scaleSize(self,scale):
      self.x = self.x * scale
      self.y = self.y * scale
```

DEFINING A CLASS

```
class Shape:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    description = ""
    author = ""
    def area(self):
        return self.x * self.y
    def perimeter(self):
        return 2 * self.x + 2 * self.y
    def describe(self,text):
        self.description = text
    def authorName(self,text):
        self.author = text
    def scaleSize(self,scale):
      self.x = self.x * scale
      self.y = self.y * scale
```

- What you have created is a description of a shape
- You have not made an actual shape
- The shape has a width (x), a height (y), and an area and perimeter (area(self) and perimeter(self))

- The function ___init___ is run when we create an instance of Shape (an actual shape, as opposed to the 'blueprint' we have here)

DEFINING A CLASS

```
class Shape:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    description = ""
    author = ""
    def area(self):
        return self.x * self.y
    def perimeter(self):
        return 2 * self.x + 2 * self.y
    def describe(self,text):
        self.description = text
    def authorName(self,text):
        self.author = text
    def scaleSize(self,scale):
      self.x = self.x * scale
      self.y = self.y * scale
```

- self:
  - how we refer to things in the class from within itself
  - the first parameter in any function defined inside a class.
  - Any function or variable created on the first level of indentation is automatically put into self
  - To access these functions and variables elsewhere inside the class, their name must be preceded with self and a full-stop (e.g. self.variable_name)

USING A CLASS

- To use a class, we need to create an actual object from the blueprint
- We call this an instance

- E.g.: `rectangle = Shape(100,45)`

- We create an instance of a class by:
  - Giving its name (in this case, rectangle)
  - Telling it what class to base itself on (Shape)
  - Putting in parentheses the values to pass to the __init__ function.
  - The init function runs (using the parameters you gave it) and returns an instance of that class, assigned to the given name

USING A CLASS

- Our class instance acts as a self-contained collection of variables and functions.
- When defining the function, we used self to access functions and variables of the class instance from within itself
- Now we use the name that we assigned to it (rectangle) to access functions and variables of the class instance from outside of itself

- #finding the area of your rectangle:
  - `print rectangle.area()`
- #finding the perimeter of your rectangle:
  - `print rectangle.perimeter()`
- #describing the rectangle
  - `rectangle.describe("A wide rectangle")`
- #making the rectangle 50% smaller
  - `rectangle.scaleSize(0.5)`
- #re-printing the new area of the rectangle
  - `print rectangle.area()`

USING A CLASS

- You can create lots of instances of the same class:
  - `longrectangle = Shape(120,10)`
  - `fatrectangle = Shape(130,120)`

- Both longrectangle and fatrectangle have their own functions and variables contained inside them - they are totally independent of each other.

- There is no limit to the number of instances you could create.

TERMINOLOGY RECAP

- Object-oriented-programming has a set of lingo that is associated with it:
  - **Classes** group together attributes and methods, so that both the data and the code to process it is in the same spot.
  - We can create any number of **instances** of that class, so that we don't have to write new code for every new object we create.
  - When we first describe a class, we are **defining** it (like with functions)
  - A variable inside a class is known as an **Attribute**
  - A function inside a class is known as a **method**
  - A class is in the same category of things as variables, lists, dictionaries, etc. That is, they are **objects**

INHERITANCE

- But what about making a fancier version of an existing class?
- Python uses a method called “inheritance”
- We define a new class, based on another, 'parent' class.
- Our new class brings everything over from the parent, and we can also add other things to it.
- If any new attributes or methods have the same name as an attribute or method in our parent class, it is used instead of the parent one.

INHERITANCE

```
class Shape:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    description = ""
    author = ""
    def area(self):
        return self.x * self.y
    def perimeter(self):
        return 2 * self.x + 2 * self.y
    def describe(self,text):
        self.description = text
    def authorName(self,text):
        self.author = text
    def scaleSize(self,scale):
      self.x = self.x * scale
      self.y = self.y * scale
```

- Let's create a child class of "Shape," called Square:

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

It is just like normally defining a class, but this time we make the parent class a parameter. We inherit everything from the parent class, and change only what needs to be.

In this case we redefined the __init__ function of Shape so that the X and Y values are the same.

INHERITANCEPTION

```
class Shape:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    description = ""
    author = ""
    def area(self):
        return self.x * self.y
    def perimeter(self):
        return 2 * self.x + 2 * self.y
    def describe(self,text):
        self.description = text
    def authorName(self,text):
        self.author = text
    def scaleSize(self,scale):
      self.x = self.x * scale
      self.y = self.y * scale

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

- Let's create another new class, this time inherited from Square. It will be two squares stuck together side by side.

```
# The shape looks like this:
#  ___________
#|     |     |
#|     |     |
#|_____|_____|

class DoubleSquare(Square):
    def __init__(self,y):
        self.x = 2 * y
        self.y = y
    def perimeter(self):
        return 2*self.x + 3*self.y
```

DEMO: MAKE A DOG

GUIDED PRACTICE: MAKE A DOG

DIRECTIONS

1. Define a class called "dog"
2. Define the __init__ function with 3 attributes
3. Define two attributes that can be changed later
4. Define five functions

DELIVERABLE

A class “dog” that can be initialized as “bulldog”

INDEPENDENT PRACTICE

MAKE A NEW DOG

ACTIVITY: INDEPENDENT PRACTICE

DIRECTIONS

1. Make a child of parent class “dog”
2. Suggestions include “smallDog” or “bigDog”
3. Change at least one feature in the __init__ function, add an attribute, and modify one function
4. Include documentation for how to create an instance of this class

DELIVERABLE

Send the code for your child class to another student and have them make an instance

CONCLUSION

PYTHON CLASSES

PYTHON CLASSES

- Classes let you define a **type** of object
- Classes are made of attributes (descriptions) and methods (functions)
- To create actual objects (instances), you need to initialize them
- Classes have inheritance, which means you can create a new class based on an existing one without recreating everything

CITATIONS

- http://sthurlow.com/python/lesson08/
- https://jeffknupp.com/blog/2014/06/18/improve-your-python-python-classes-and-object-oriented-programming/
- Python documentation: https://docs.python.org/3/tutorial/classes.html
- Codecademy Practice (Unit 11: Classes): https://www.codecademy.com/learn/python