Class 8

[w200] MIDS Python Bridge Course Spring 2018

Course Content | moving into **OOP**

Unit 1 | Introduction, the Command Line, Source Control

Unit 2 | Starting Out with Python

Unit 3 | Sequence Types and Dictionaries

Unit 4 | More About Control and Algorithms

Unit 5 | Functions

Unit 6 | Modules and Packages

Unit 7 | Classes

Unit 8 | Object-Oriented Programming

Agenda

Homework Review

Midterm (10% of final grade)

Inheritance and Polymorphism

Breakout

Midterm Review

Project 1 Discussion

Homework 6 - Scrabble Grading

- Overall: Really nice (and varied) Scrabble Implementations
- Good Algorithm design:
    - Iterated through scrabble words list only once
    - For wildcards: Did NOT make a list of all possible rack letters
- Compared words to rack while keeping track of wildcards:
    - If word length > rack length - immediately discard (& move to next word)
    - If letter not in rack & no wildcards - discard
        1. If wildcards, word could be 'off' by that many letters (1 or 2)
        2. If word was off by more than # of wildcards - discard
    - If word passes above tests - add to valid word list & go to next word

Homework 6 - Top Questions

- Tabs v Spaces
  - Eternal debate!
  - A reason 4 spaces are better: 4 spaces = 4 spaces wherever you copy and paste the code to; 1 tab isn’t necessarily 4 spaces (tab sizes are different, space size is not)
- 79 Character Limit
  - A little arbitrary today (with wide-screen monitors etc)
  - Comes from when UNIX default windows were 80 characters wide
  - Designed to increase readability of code
  - Jupyter nb does not wrap code so code could continue off the screen to the right

Homework 6 - Top Questions

- Docstings
    - Usually done after defining a function to indicate the input arguments and type, what the function does and what the function returns; Ex:

```
def function_with_docstring(param1, param2):
    """This function takes a string, param2, and checks if a number, param1,
    is in that string.

    Args:
        param1 (int): The number to check for
        param2 (str): The string to check

    Returns:
        bool: True for success, False otherwise.
    """
```

Homework 6 - Top Questions

- Autoformat PEP8?
    - Yes in PyCharm (and maybe others)
    - Might be better in the long run just to start coding that way (in case the tool isn't available)

- Dunder is the __ (like in __init__)
    - Invoked 'behind the scenes' - that is, you don't specifically call that method
    - Using __init__ as an example - its a method that is automatically called when you first substantiate that object
- Public vs private variables
    - Private Variables are annotated by a dundar __ or one _ (like __count)
    - Means: that variable is only accessed by that class not from outside that clas
    - Python doesn't 'truly' have private variables - can still modify these
      (in other languages, private variables cannot be modified outside of the class)

Assignment Review | Week 7

Due date moved back one week:

- Mon, March 5th for Tuesday classes
- Wed, March 7th for Thursday classes

Questions & Problem Areas?

Assignment Review | Project

Reminder: Your proposal was due, but treat it as your **hypothesis** as to how you'll solve the problem. You have **two weeks** to implement, refine, and adjust!

Next week you will have time to discuss your progress and outstanding issues with your breakout group.

Reminders |

Course Schedule

https://docs.google.com/spreadsheets/d/1Skg_b0rM5jPcVg0ixGrPnK5-QCGrHaFVr1afgchUN5c

Agenda

Homework Review

Midterm (10% of final grade)

Inheritance and Polymorphism

Breakout

Midterm Review

Project 1 Discussion

Agenda

Homework Review

Midterm (10% of final grade)

Inheritance and Polymorphism

Breakout

Midterm Review

Project 1 Discussion

Classes, Inheritance, Polymorphism

Disclaimer: This stuff can be hard to wrap your head around!!!

Why Classes?

1. Encapsulation
2. Modularity
3. Inheritance
4. Polymorphism

Classes, Inheritance, Polymorphism

Why did we have you implement a “card” class, even though it didn’t need functions?

How can you decide when to use an “object” versus an “attribute” for a given part of your code?

Classes, Inheritance, Polymorphism

Discuss (Async Notebooks):

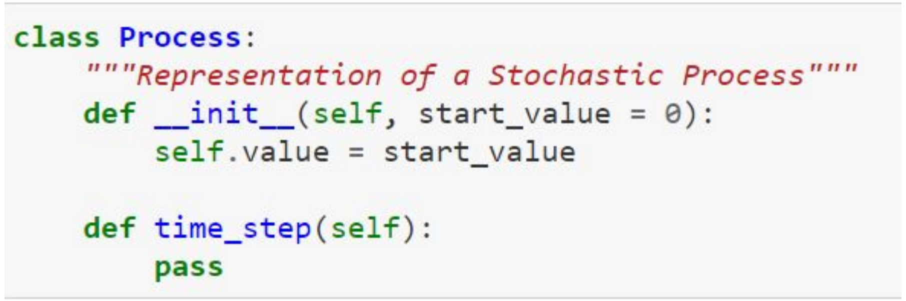
- Inheritance (8.4)
- Polymorphism (8.7)
- Magic Methods (8.9)

Classes, Inheritance, Polymorphism

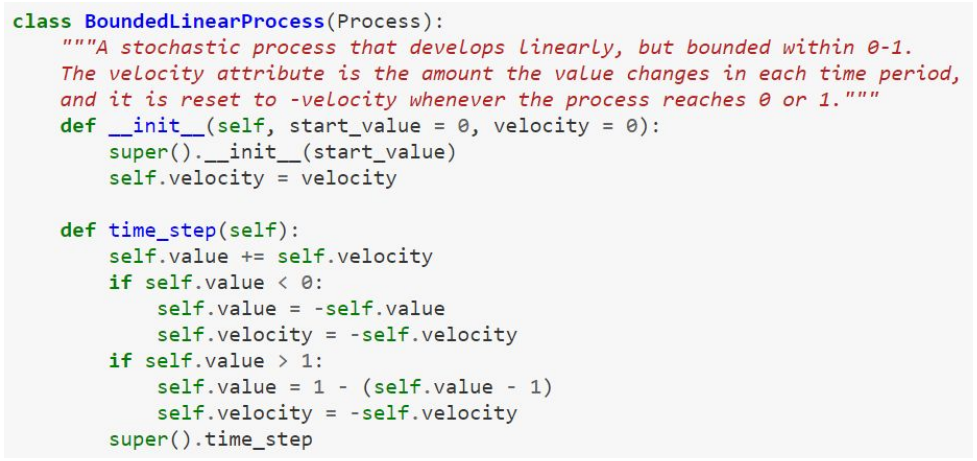
Why would we want to use inheritance?

What is “super”?

What is “pass”?



```python
class Process:
    """Representation of a Stochastic Process"""
    def __init__(self, start_value = 0):
        self.value = start_value

    def time_step(self):
        pass
```



```python
class BoundedLinearProcess(Process):
    """A stochastic process that develops linearly, but bounded within 0-1.
    The velocity attribute is the amount the value changes in each time period,
    and it is reset to -velocity whenever the process reaches 0 or 1."""
    def __init__(self, start_value = 0, velocity = 0):
        super().__init__(start_value)
        self.velocity = velocity

    def time_step(self):
        self.value += self.velocity
        if self.value < 0:
            self.value = -self.value
            self.velocity = -self.velocity
        if self.value > 1:
            self.value = 1 - (self.value - 1)
            self.velocity = -self.velocity
        super().time_step
```

Classes, Inheritance, Polymorphism

Why would we want to use inheritance?

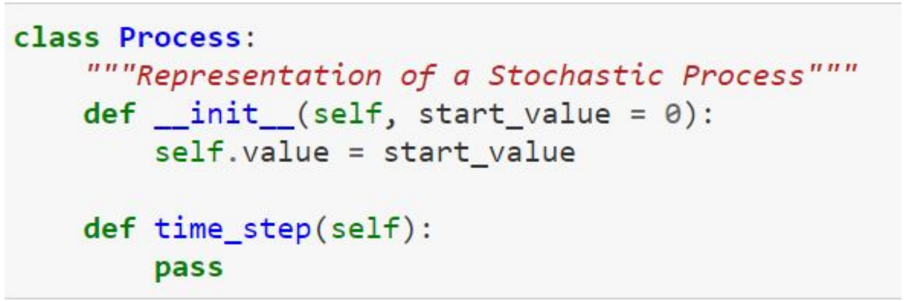
A template for other classes

What is “super”?

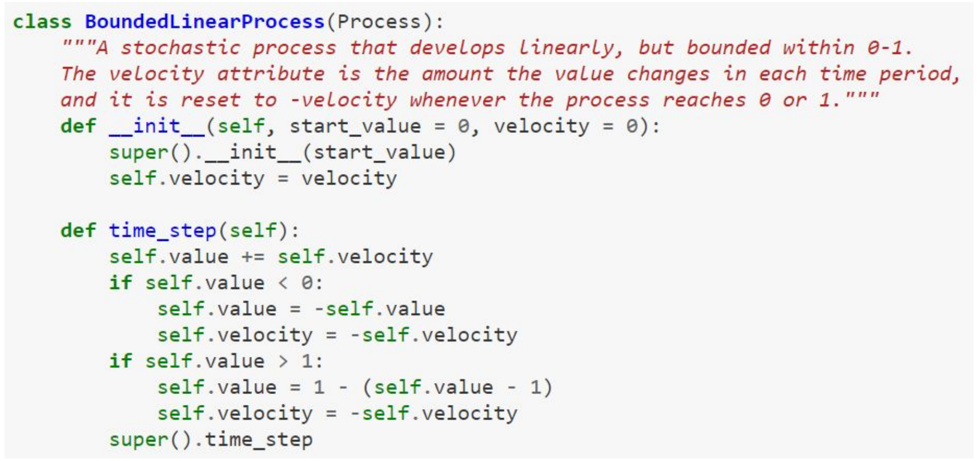
Run the function as defined in the superclass

What is “pass”?

Define a function, to be implemented in subclasses



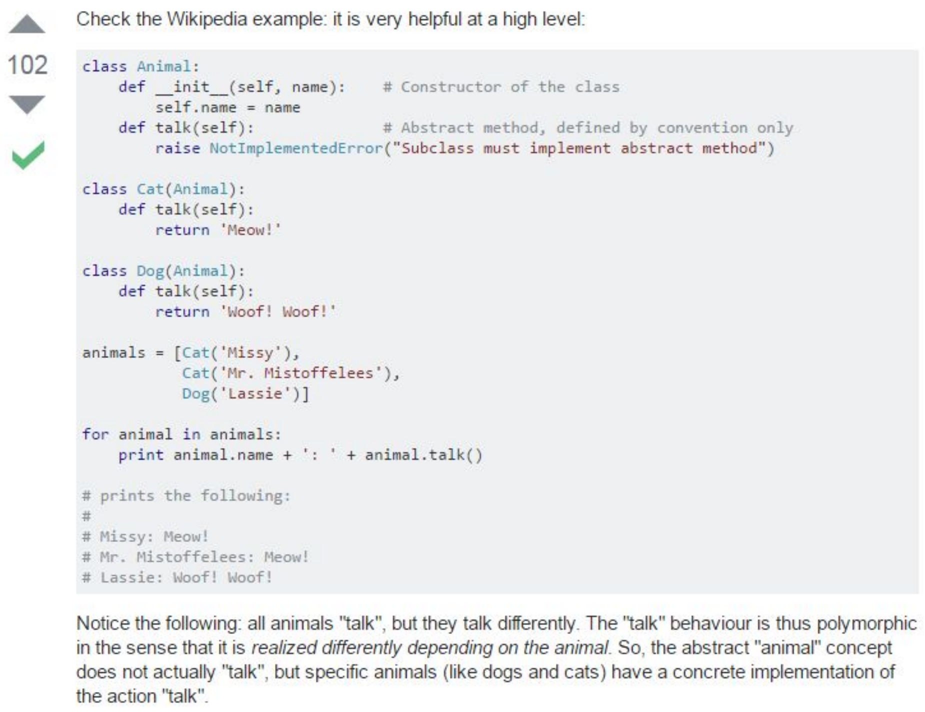
```python
class Process:
    """Representation of a Stochastic Process"""
    def __init__(self, start_value = 0):
        self.value = start_value

    def time_step(self):
        pass
```



```python
class BoundedLinearProcess(Process):
    """A stochastic process that develops linearly, but bounded within 0-1.
    The velocity attribute is the amount the value changes in each time period,
    and it is reset to -velocity whenever the process reaches 0 or 1."""
    def __init__(self, start_value = 0, velocity = 0):
        super().__init__(start_value)
        self.velocity = velocity

    def time_step(self):
        self.value += self.velocity
        if self.value < 0:
            self.value = -self.value
            self.velocity = -self.velocity
        if self.value > 1:
            self.value = 1 - (self.value - 1)
            self.velocity = -self.velocity
        super().time_step
```

Classes, Inheritance, Polymorphism

What is polymorphism?



102

Check the Wikipedia example: it is very helpful at a high level:

```
class Animal:
    def __init__(self, name):    # Constructor of the class
        self.name = name
    def talk(self):              # Abstract method, defined by convention only
        raise NotImplementedError("Subclass must implement abstract method")

class Cat(Animal):
    def talk(self):
        return 'Meow!'

class Dog(Animal):
    def talk(self):
        return 'Woof! Woof!'

animals = [Cat('Missy'),
           Cat('Mr. Mistoffelees'),
           Dog('Lassie')]

for animal in animals:
    print animal.name + ': ' + animal.talk()

# prints the following:
#
# Missy: Meow!
# Mr. Mistoffelees: Meow!
# Lassie: Woof! Woof!
```

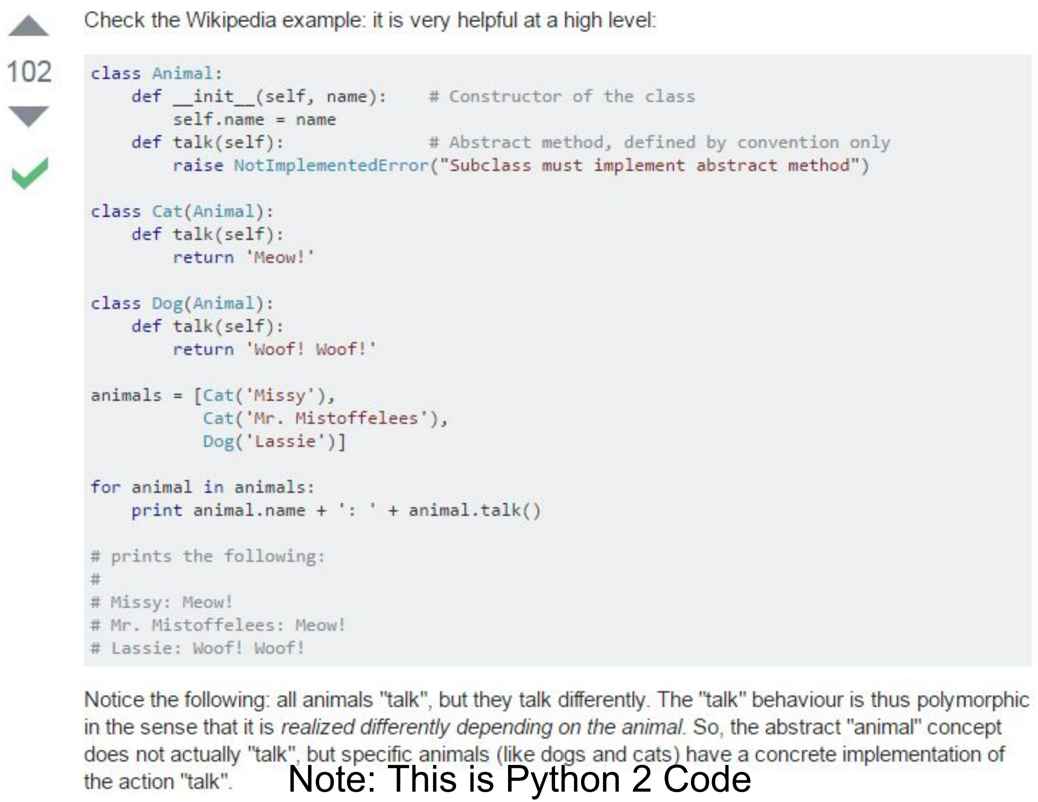
Notice the following: all animals "talk", but they talk differently. The "talk" behaviour is thus polymorphic in the sense that it is *realized differently depending on the animal*. So, the abstract "animal" concept does not actually "talk", but specific animals (like dogs and cats) have a concrete implementation of the action "talk".

Note: This is Python 2 Code

Classes, Inheritance, Polymorphism

What is polymorphism?
The provision of a single interface to entities of different types.

Discuss: How may this be related to inheritance?



Check the Wikipedia example: it is very helpful at a high level:

```
class Animal:
    def __init__(self, name):    # Constructor of the class
        self.name = name
    def talk(self):              # Abstract method, defined by convention only
        raise NotImplementedError("Subclass must implement abstract method")

class Cat(Animal):
    def talk(self):
        return 'Meow!'

class Dog(Animal):
    def talk(self):
        return 'Woof! Woof!'

animals = [Cat('Missy'),
           Cat('Mr. Mistoffelees'),
           Dog('Lassie')]

for animal in animals:
    print animal.name + ': ' + animal.talk()

# prints the following:
#
# Missy: Meow!
# Mr. Mistoffelees: Meow!
# Lassie: Woof! Woof!
```

Notice the following: all animals "talk", but they talk differently. The "talk" behaviour is thus polymorphic in the sense that it is *realized differently depending on the animal*. So, the abstract "animal" concept does not actually "talk", but specific animals (like dogs and cats) have a concrete implementation of the action "talk".

Note: This is Python 2 Code

Classes, Inheritance, Polymorphism

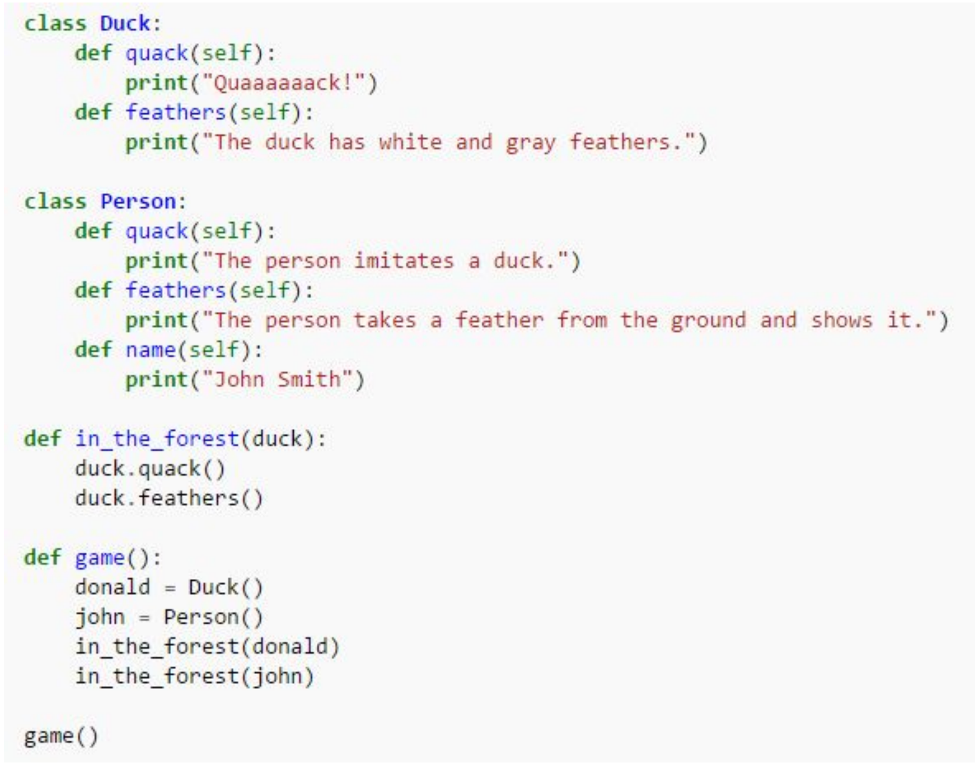
What is duck typing?

Classes, Inheritance, Polymorphism

What is duck typing?

Python will not check variable Types before running a function.

“If it looks like a duck and quacks like a duck, it is a duck!”

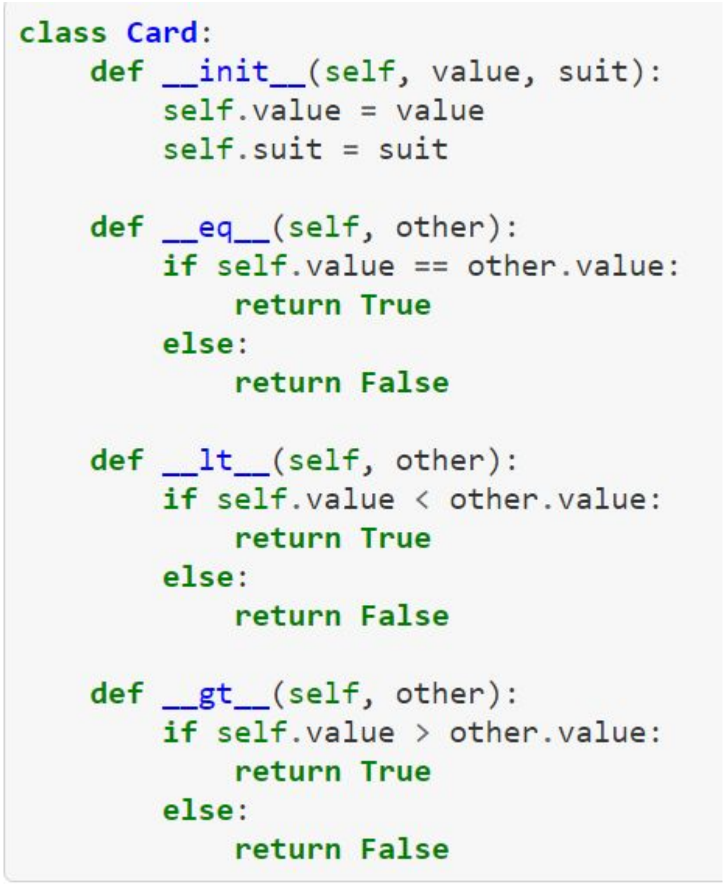


```
class Duck:
    def quack(self):
        print("Quaaaaaack!")
    def feathers(self):
        print("The duck has white and gray feathers.")

class Person:
    def quack(self):
        print("The person imitates a duck.")
    def feathers(self):
        print("The person takes a feather from the ground and shows it.")
    def name(self):
        print("John Smith")

def in_the_forest(duck):
    duck.quack()
    duck.feathers()

def game():
    donald = Duck()
    john = Person()
    in_the_forest(donald)
    in_the_forest(john)

game()
```

Magic Methods

What are magic methods?

Can you describe the purpose of the code on this slide?



```python
class Card:
    def __init__(self, value, suit):
        self.value = value
        self.suit = suit

    def __eq__(self, other):
        if self.value == other.value:
            return True
        else:
            return False

    def __lt__(self, other):
        if self.value < other.value:
            return True
        else:
            return False

    def __gt__(self, other):
        if self.value > other.value:
            return True
        else:
            return False
```

Agenda

Homework Review

Midterm (10% of final grade)

Inheritance and Polymorphism

Breakout

Midterm Review

Project 1 Discussion

Breakout

Let’s adopt a pet...

Gamify | Breakout - Allow your pets to interact

Agenda

Homework Review

Midterm (10% of final grade)

Inheritance and Polymorphism

Breakout

Midterm Review

Project 1 Discussion

Midterm Review

Live Q & A using Poll Features

Agenda

Homework Review

Midterm (10% of final grade)

Inheritance and Polymorphism

Breakout

Midterm Review

Project 1 Discussion

The Project | Your Mission

Create a small, object-oriented program of your choosing:

Examples:

- An ATM
- A flower shop
- An adventure game
- Something relating to your everyday work

The Project | Code

Python 3 code, 300-500 lines (750 max)

All code should be well commented!

Must use Object Oriented design and classes

Demonstrate various flow controls and data types

Robust to common user errors and exceptions

The Project | Your Mission

The user will interact with your program via Terminal/Shell

Three documents due before your due date class:

1. Proposal (10%)
2. Code(s) (80%)
3. Reflective Summary (10%)

You will demo your progress in a breakout room one week before the presentation

You may only use Python libraries that come installed with Anaconda

The Project | Proposal

Describe your project concept

Pseudocode your major classes and functions

1. Briefly describe the purpose of each class
2. List expected functions belong to each class
3. List inputs and outputs for each function

Instructors will “approve” your draft proposal

Coding is **iterative**. Your final code may not match the proposal exactly

The Project | Reflection

Submit a 1-page reflection with your code

Instructors will read your reflection before grading your project

Tell us how to use your project!

Discuss challenges you faced and how you overcame them

The Project | Demo

As time allows, show 1-2 examples of strong projects from last semester.

The Project | Questions

Up next | Numpy, strings, Data analysis

Project 1 | build your own object oriented project

- Code at home and collaborate in class

Unit 9 | Working With Text and Binary Data

Unit 10 | NumPy

Unit 11 | Data Analysis With Pandas

Unit 12 | More Analysis With Pandas

Unit 13 | Testing