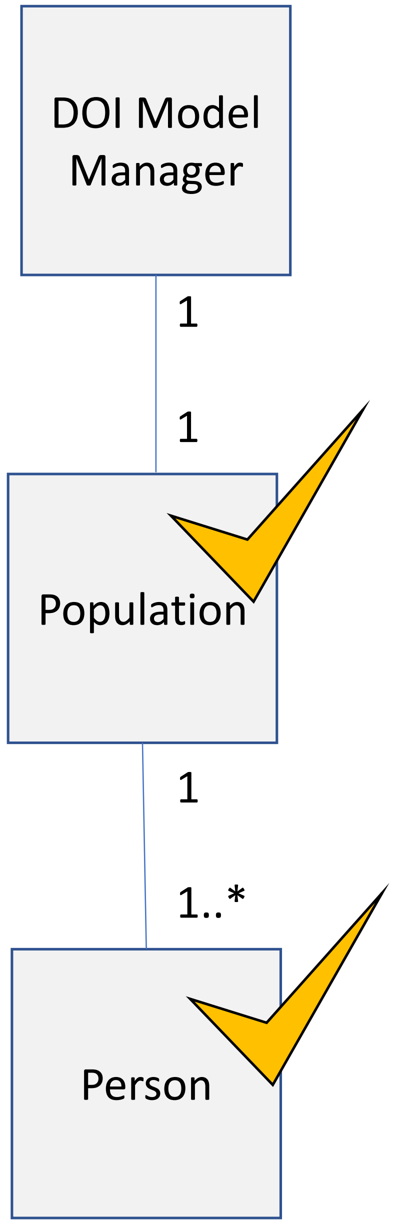
04-5 Python Application Diffusion of Innovation Modeling

CSI 500

Modeling Diffusion of Innovation in Python

- Recap: we've coded up the Person and Population classes
- Let's now move to the actual Model class



Let's build the DOI model

- We import matplotlib for plots
- We import our Diffusion package for Population class
- We create a DOI_Model class to manage the model
- __init__ sets up our history lists, model parameters, and initial Population object

```
import matplotlib.pyplot as plt
import Diffusion as df

class DOI_Model(object):
    def __init__( self, N=500, beta=0.09, \
              gamma=0.01, max_time=250 ):

        self.potential_history = []
        self.adoption_history = []
        self.disposal_history = []

        self.N = N
        self.beta = beta
        self.gamma = gamma
        self.max_time = max_time
        self.timespan = range(max_time)

        self.pop = df.Population(N, beta, gamma)
```

Building the Model __str__

- we'll add the obligatory __str__ method to enable printing

```
def __str__( self ):
    msg = 'DOI_Model: %d, %8.2f, %8.2f, %d' % \
    (self.N, self.beta, self.gamma, self.max_time)
    return msg
```

Running the model

- Here's the run() method
- We just iterate over the timespan
- at each time step, we count and save the number of potentials, adopters, and disposers
- we then model adoption and disposal
- that's it!

```
def run(self):
    for time in self.timespan:

        # keep track of what happened
        self.potential_history.append(self.pop.num_potentials)
        self.adoption_history.append(self.pop.num_adopters)
        self.disposal_history.append(self.pop.num_disposers)

        # model population actions
        self.pop.model_adoption()
        self.pop.model_disposal()
```

Print pretty graphics

- The plot() method prints out the counts of each category observed at each time step
- note the use of blue, red, and green colors
- note the use of labels for each plotted line
- observe title, xlabel and ylabel
- observe use of legend

```
def plot(self):
    #
    # plot results
    #
    plt.plot(self.timespan, self.potential_history, '-b', label="potential")
    plt.plot(self.timespan, self.adoption_history, '-r', label="adopters")
    plt.plot(self.timespan, self.disposal_history, '-g', label="disposers")

    plt.title('diffusion of innovation with random mixing')
    plt.xlabel('time')
    plt.ylabel('adoption rate')

    plt.ylim(0, pop.N)
    plt.legend(title="key", loc='center right')

    plt.show()
```

A bit more Python Packaging

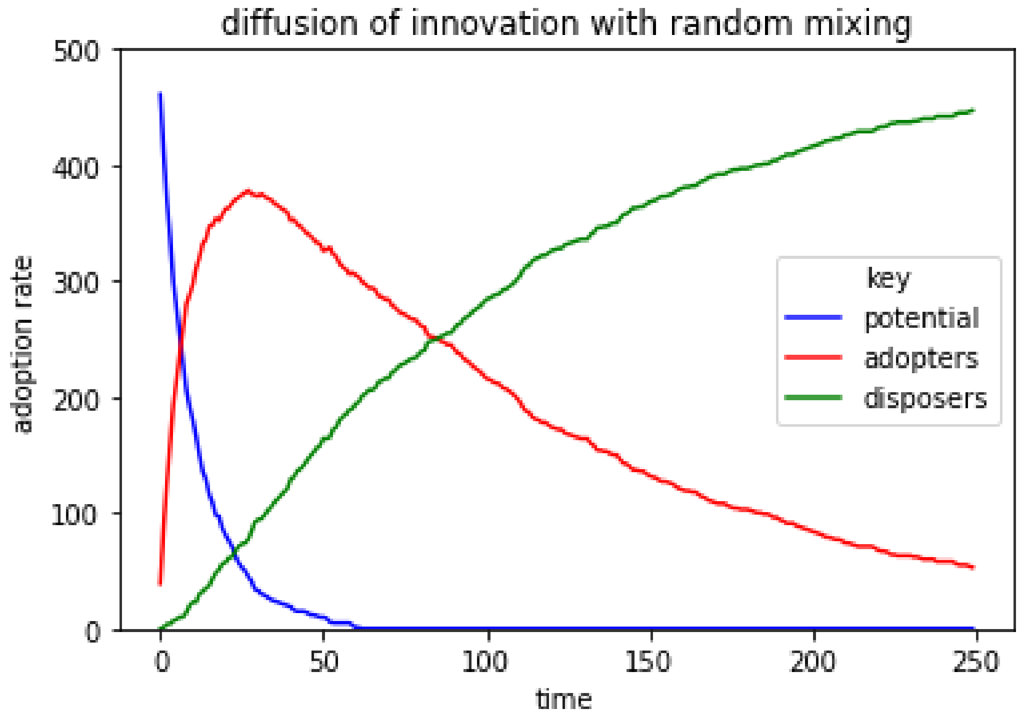
- We can put the DOI_Model class we wrote into a file with a .py extension, such as "DOI Model.py"
- Note that the previous package we created, "Diffusion.py", needs to be in the same folder as the DOI Model.py file we wrote here
- Note we need to declare a main() that instantiates, runs, and plots a DOI_Model

DOI Model.py

```
import Diffusion
import matplotlib.pyplot as plt

class DOI_Model

#
# object-oriented main
#
def main():
    model = DOI_Model(N=500, beta=0.09, \
                gamma=0.02, max_time = 250)
    model.run()
    model.plot()

#
# use standard Python idiom
#
if __name__ == '__main__':
    main()
```

Here's what it looks like

- The completed model looks like this when run
- Note the quick drop-off among potentials
- Note the steady rise of disposers
- Note the "bubble" of adopters - this is a very idiomatic diffusion of innovation feature



Summary

- Before coding, analyze the problem space
  - block diagrams
  - finite state machines
  - class models
  - high-level functional specifications
- Python classes used to create a diffusion of innovation model
  - Person class for a person
  - Population class for a population
  - DOI_Model class manages the simulation
- Object oriented design becomes very straightforward
  - easy to change code as needed
  - code localized to specific object types