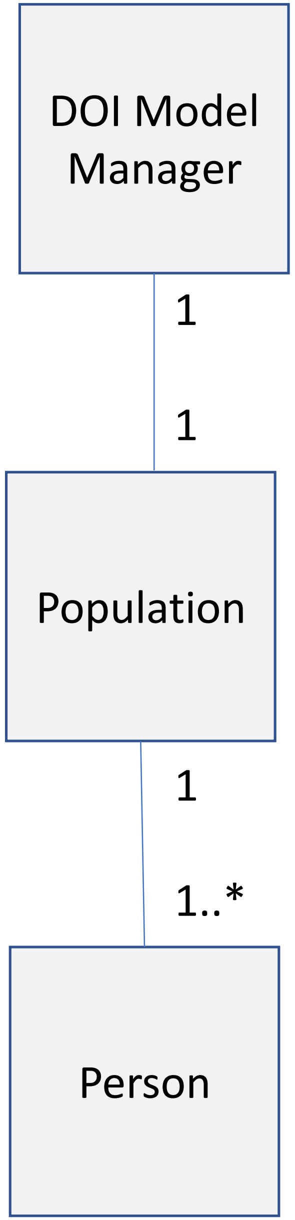
04-4 Python Application Diffusion of Innovation Development

CSI 500

Modeling Diffusion of Innovation in Python

- Recap: we've done our initial analysis of the problem space
  - Developed a good understanding of the conceptual design
  - Built Finite-State-Machine model, UML class model
  - Written high-level functional specfications
- Now let's focus on building the foundational classes for our model
  - Population to model a group of Persons
  - Person to model the individuals



The Person Class

- This class holds our representation of a person in a Diffusion of Innovation model
  - we'll need to use the numpy library
  - status is used to track of which category they reside in : Potential, Adopter, or Disposer
  - pid is an ID number just for bookkeeping
- Define mandatory __init__ and __str__ methods
- Add some helper methods to change categories

```
#
# Define Person
#
import numpy as np
class Person( object ):

    def __init__( self, pid=0 ):
        self.pid = pid
        self.status = "Potential"

    def __str__( self ):
        msg = '%d,%s' % \
            ( self.pid, self.status )
        return msg

    def adopt(self):
        self.status = "Adopt"

    def dispose(self):
        self.status = "Dispose"
```

The Population Class __init__

- This class holds our representation of a population of persons in a Diffusion of Innovation model
  - Let's start with the mandatory __init__ method
  - Define some default parameters
  - Define some instance variables

```
#
# define a Population
#
import numpy as np
class Population( object ):
    """
    simulate a population of Persons
    in a 3-compartment diffusion of innovation model
    """

    def __init__( self, N=100, beta=0.05, gamma=0.03 ):
        self.N = N
        self.beta = beta
        self.gamma = gamma

        self.num_potentials = self.N
        self.num_adopters = 0
        self.num_disposers = 0

        self.person_list = []
        self.setup(  )
```

The Population Class __str__

- Now let's add the mandatory __str__ method
  - lets us print out details of a Population using print()

```
def __str__( self ):
    msg = " "
    msg = ''%d,%g,%g,%g,%g,%g' % \
      ( self.N,
       self.beta,
       self.gamma,
       self.num_potentials,
       self.num_adopters,
       self.num_disposers )
    return msg
```

The Population Class: setup

- Let's look at how to initialize the Population class
  - We'll iterate from 0 to N-1
  - create a new Person with a pid
  - we'll initialize with approximately beta*N adopters
  - the rest will be considered potentials
  - We keep track of the individual Persons by putting them in the person_list

```
def setup( self ):
    for i in range( self.N ):
        p = Person( pid=i )
        chance = np.random.random(  )
        if chance < self.beta:
            p.adopt(  )
            self.num_potentials -= 1
            self.num_adopters += 1
        self.person_list.append( p )
```

The Population Class: adoption

- Let's look at how to model adoption
  - We'll iterate from 0 to N-1
  - if the i-th Person is a Potential
    - then we simulate via chance that they switch to become an adopter
    - the rate is governed by beta
  - If they switch, we adjust of the total number of potentials and adopters so the bookkeeping is straight

```
def model_adoption( self ):
    for i in range( self.N ):
        p = self.person_list[i]
        if ( p.status == "Potential" ):
            chance = np.random.random(  )
            if chance < self.beta:
                p.adopt(  )
                self.num_potentials -= 1
                self.num_adopters += 1
```

The Population Class: dispose

- Let's look at how to model disposal
  - We'll iterate from 0 to N-1
  - if the i-th Person is an Adopter
    - then we simulate via chance that they switch to become a Disposer
    - the rate is governed by gamma
  - If they switch, we adjust of the total number of adopters and disposers so the bookkeeping is straight

```
def model_disposal( self ):
    for i in range( self.N ):
        p = self.person_list[i]
        if ( p.status == "Adopt" ):
            chance = np.random.random(  )
            if chance < self.gamma:
                p.dispose(  )
                self.num_adopters -= 1
                self.num_disposers += 1
```

A bit of Python Packaging

- We can take the Person and Population code we just wrote and put it into a Python "package" file
  - For example, call it "Diffusion.py"
- That's a file with a .py extension containing classes, methods, and constants
- Here's what it would look like
- Later we can use it by just typing in "import Diffusion" in our model code!

Diffusion.py

class Person

class Population

```
# include standard Python package blurb
def main():
    pass

if __name__ == '__main__':
    main()
```

Summary

- Python classes used to create key parts of the diffusion of innovation model
  - Person class for a person
  - Population class for a population
  - DOI_Model class manages the simulation will be developed next
- Python "package" contains classes, methods, and constants
  - used to organize Python software projects
  - Easy to invoke, just use "import <packagename>"