PART B - PROBLEM 2: IMPLEMENTING A SIMPLE SIMULATION (NO DRUG TREATMENT)

(15/15 points)

We start with a trivial model of the virus population - the patient does not take any drugs and the viruses do not acquire resistance to drugs. We simply model the virus population inside a patient as if it were left untreated.

SIMPLEVIRUS CLASS

To implement this model, you will need to fill in the `SimpleVirus` class, which maintains the state of a single virus particle. You will implement the methods `__init__`, `getMaxBirthProb`, `getClearProb`, `doesClear`, and `reproduce` according to the specifications. Use `random.random()` for generating random numbers to ensure that your results are consistent with ours.

Hint: random seed

The `reproduce` method in `SimpleVirus` should produce an offspring by returning a new instance of `SimpleVirus` with probability: **self.maxBirthProb * (1 - popDensity)**. This method raises a `NoChildException` if the virus particle does not reproduce. For a reminder on raising execptions, review the Python docs.

`self.maxBirthProb` is the birth rate under optimal conditions (the virus population is negligible relative to the available host cells so there is ample nourishment available). `popDensity` is defined as the ratio of the current virus population to the maximum virus population for a patient and should be calculated in the `update` method of the `Patient` class.

PATIENT CLASS

You will also need to implement the `Patient` class, which maintains the state of a virus population associated with a patient.

The `update` method in the `Patient` class is the inner loop of the simulation. It modifies the state of the virus population for a single time step and returns the total virus population at the end of the time step. At every time step of the simulation, each virus particle has a fixed probability of being cleared (eliminated from the patient's body). If the virus particle is not cleared, it is considered for reproduction. The virus population should never exceed `maxPop`; if you utilize the population density correctly, you shouldn't need to provide an explicit check for this.

Unlike the clearance probability, which is constant, the probability of a virus particle reproducing is a function of the virus population. With a larger virus population, there are fewer resources in the patient's body to facilitate reproduction, and the probability of reproduction will be lower. One way to think of this limitation is to consider that virus particles need to make use of a patient's cells to reproduce; they cannot reproduce on their own. As the virus population increases, there will be fewer available host cells for viruses to utilize for reproduction.

To summarize, `update` should first decide which virus particles are cleared and which survive by making use of the `doesClear` method of each `SimpleVirus` instance, then update the collection of `SimpleVirus` instances accordingly. With the surviving `SimpleVirus` instances, `update` should then call the `reproduce` method for each virus particle. Based on the population density of the surviving `SimpleVirus` instances, `reproduce` should either return a new instance of `SimpleVirus` representing the offspring of the virus particle, or raise a `NoChildException` indicating that the virus particle does not reproduce during the current time step. The `update` method should update the attributes of the patient appropriately under either of these conditions. After iterating through all the virus particles, the `update` method returns the number of virus particles in the patient at the end of the time step.

Hint: mutating objects

Note that the mapping between time steps and actual time will vary depending on the type of virus being considered, but for this problem set, think of a time step as a simulated hour of time.

About the grader: When defining a `Patient` class member variable to store the viruses list representing the virus population, please use the name `self.viruses` in order for your code to be compatible with the grader and to pass one of the tests.

```
# Enter your definitions for the SimpleVirus and Patient classes in this box.
#
# PROBLEM 2
#
class SimpleVirus(object):

    """
    Representation of a simple virus (does not model drug effects/resistance).
    """
    def __init__(self, maxBirthProb, clearProb):
        """
        Initialize a SimpleVirus instance, saves all parameters as attributes
        of the instance.
        maxBirthProb: Maximum reproduction probability (a float between 0-1)
        clearProb: Maximum clearance probability (a float between 0-1).
        """
```

Correct

Test results

Hide output

CORRECT

Test: Initialization 1

Initialize a SimpleVirus

Output:

Passed test

Test: Initialization 2

Initialize a Patient

Output:

Passed test

Test: SimpleVirus 1

Initialize a SimpleVirus that is never cleared and always reproduces

Output:

```
v1 = SimpleVirus(1.0, 0.0)
Testing 'doesClear' and 'reproduce' methods
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
Test completed.
```

`Test: SimpleVirus 2`

Initialize a SimpleVirus that is never cleared and never reproduces

Output:

```
v1 = SimpleVirus(0.0, 0.0)
Testing 'doesClear' and 'reproduce' methods
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
v1.doesClear(): False
Test completed.
```

`Test: SimpleVirus 3`

Initialize a SimpleVirus that is always cleared and always reproduces

Output:

```
v1 = SimpleVirus(1.0, 1.0)
Testing 'doesClear' and 'reproduce' methods
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
Test completed.
```

`Test: SimpleVirus 4`

Initialize a SimpleVirus that is always cleared and never reproduces

Output:

```
v1 = SimpleVirus(0.0, 1.0)
Testing 'doesClear' and 'reproduce' methods
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
v1.doesClear(): True
Test completed.
```

Test: SimpleVirus 5

Initialize a SimpleVirus with randomized probabilities

Output:

```
v1 = SimpleVirus(0.98, 0.58)
Testing reproduce. Be sure your implementation is making EXACTLY ONE call to random.random(),
and no other random module calls.
popDensity = 0.03
Reproduced successfully
Reproduced successfully
Reproduced successfully
Reproduced successfully
Reproduced successfully
Reproduced successfully
Reproduced successfully
Reproduced successfully
Reproduced successfully
Raised 'NoChildException'
Test completed.
```

Test: class Patient 1

Initialize a Patient with randomized viruses

Output:

```
viruses = [ SimpleVirus(0.51, 0.54) SimpleVirus(0.4, 0.28) SimpleVirus(0.49, 0.22)
SimpleVirus(0.02, 0.43) SimpleVirus(0.46, 0.06) ]
P1 = Patient(viruses, 6)
P1.getTotalPop() = 5
```

Test: class Patient 2

Create a Patient with virus that is never cleared and always reproduces

Output:

```
virus = SimpleVirus(1.0, 0.0)
patient = Patient([virus], 100)
Updating the patient for 100 trials...
patient.getTotalPop() expected to be >= 100
Test successfully completed
```

Test: class Patient 3

Create a Patient with virus that is always cleared and always reproduces

Output:

```
virus = SimpleVirus(1.0, 1.0)
patient = Patient([virus], 100)
Updating the patient for 100 trials...
patient.getTotalPop() expected to = 0
Test successfully completed
```

Test: class Patient 4

Initialize a Patient with randomized viruses

Output:

```
viruses = [ SimpleVirus(0.52, 0.81) SimpleVirus(0.59, 0.97) SimpleVirus(0.06, 0.97)
SimpleVirus(0.58, 0.91) SimpleVirus(0.1, 0.98) ]
P1 = Patient(viruses, 8)
P1.getTotalPop() = 5
Updating patient 10 times... all exceptions should be handled...
len(P1.viruses) < maxPop? True
len(P1.viruses) < maxPop? True
len(P1.viruses) < maxPop? True
len(P1.viruses) < maxPop? True
len(P1.viruses) < maxPop? True
len(P1.viruses) < maxPop? True
len(P1.viruses) < maxPop? True
len(P1.viruses) < maxPop? True
len(P1.viruses) < maxPop? True
len(P1.viruses) < maxPop? True
Test Completed
```

Test: class Patient 5

Check exception handling by raising different types of exceptions in SimpleVirus.reproduce

Output:

```
Successfully ignored raised exception of type: AttributeError
Successfully ignored raised exception of type: ZeroDivisionError
Successfully ignored raised exception of type: TypeError
Successfully ignored raised exception of type: ValueError
Successfully ignored raised exception of type: NameError
Successfully caught raised 'NoChildException'
Test Completed
```

Hide output

Check Save *You have used 3 of 30 submissions*

Show Discussion

New Post

EdX offers interactive online classes and MOOCs from the world's best universities. Online courses from MITx, HarvardX, BerkeleyX, UTx and many other universities. Topics include biology, business, chemistry, computer science, economics, finance, electronics, engineering, food and nutrition, history, humanities, law, literature, math, medicine, music, philosophy, physics, science, statistics and more. EdX is a non-profit online initiative created by founding partners Harvard and MIT.

Terms of Service and Honor Code

Privacy Policy (Revised 4/16/2014)

About & Company Info

About

News

Contact

FAQ

edX Blog

Donate to edX

Jobs at edX

Follow Us

Twitter

Facebook

Meetup

LinkedIn

Google+