Lakshmi Narain College Of Technology Excellence

Bhopal

Computer Science

&

Training And Placement Department

Topic : Smart Health Prediction System

Submitted By : Prince Dwivedi

Submitted To : Praveen Sir

Enrollment Number : 0176CS191122

Year : 2nd (3rd Semester)

Introduction

Using Python and Pylab, designed and implemented a stochastic simulation of patient and virus population dynamics, and reached conclusions about treatment regimens based on the simulation results.

Background: Viruses, Drug Treatments, and Computational Models

Viruses such as HIV and H1N1 represent a significant challenge to modern medicine. One of the reasons that they are so difficult to treat is their ability to evolve.

As you may know, the traits of an organism are determined by its genetic code. When organisms reproduce, their offspring will inherit genetic information from their parent. This genetic information will be modified, either because of mixing of the two parents' genetic information, or through mutations in the genome replication process, thus introducing diversity into a population.

Viruses are no exception. Two characteristics of viruses make them particularly difficult to treat. The first is that their replication mechanism often lacks the error checking mechanisms that are present in more complex organisms. This speeds up the rate of mutation. Secondly, viruses replicate extremely quickly (orders of magnitude faster than humans) -- thus, while we may be used to thinking of evolution as a process which occurs over long time scales, populations of viruses can undergo substantial evolutionary changes within a single patient over the course of treatment.

These two characteristics allow a virus population to acquire genetic resistance to therapy quickly. In this problem set, we will make use of simulations to explore the effect of introducing drugs on the virus population and determine how best to address these treatment challenges within a simplified model.

Computational modeling has played an important role in the study of viruses such as HIV (for example, see this paper, by MIT graduate David Ho). We will implemented a highly simplified stochastic model of virus population dynamics. Many details have been swept under the rug (host cells are not explicitly modeled and the size of the population is several orders of magnitude less than the size of actual virus populations). Nevertheless, our model exhibits biologically relevant characteristics and will give you a chance to analyze and interpret interesting simulation data.

Spread of a Virus in a Person

In reality, diseases are caused by viruses and have to be treated with medicine, so we'll be looking at a detailed simulation of the spread of a virus within a person .

Implementing a Simple Simulation (No Drug Treatment)

Started 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

For implementing this model, I made `SimpleVirus` class, which maintains the state of a single virus particle and implemented following methods : `__init__`, `getMaxBirthProb`, `getClearProb`,`doesClear`, and `reproduce` .

The `reproduce` method in `SimpleVirus` produces 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.

`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.

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

E:\PY FILES\ProblemSet3\ProblemSet3

Editor - E:\PY FILES\ProblemSet3\ProblemSet3\LNCT_PROJECT.py

cecows.py | lam.py | lab0.py | ps4.py | linear_q.py | edxfingers.py | ps3b.py | LNCT_PROJECT.py

```python
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).
        """
        self.maxBirthProb=maxBirthProb
        self.clearProb=clearProb


    def getMaxBirthProb(self):
        """
        Returns the max birth probability.
        """
        return self.maxBirthProb


    def getClearProb(self):
        """
        Returns the clear probability.
        """
        return self.clearProb



    def doesClear(self):
        """ Stochastically determines whether this virus particle is cleared from the
        patient's body at a time step.
        returns: True with probability self.getClearProb and otherwise returns
        False.
        """
```

IPython console

Console 1/A

In [16]:

Permissions: RW | End-of-lines: CRLF | Encoding: ASCII | Line: 571 | Column: 1 | Memory: 87 %

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

E:\PY FILES\ProblemSet3\ProblemSet3

Editor - E:\PY FILES\ProblemSet3\ProblemSet3\LNCT_PROJECT.py

cecows.py | lam.py | lab0.py | ps4.py | linear_q.py | edxfingers.py | ps3b.py | LNCT_PROJECT.py

```python


        prob = random.random()
        if prob <= self.clearProb:
            return True
        else:
            return False

    def reproduce(self, popDensity):
        """
        Stochastically determines whether this virus particle reproduces at a
        time step. Called by the update() method in the Patient and
        TreatedPatient classes. The virus particle reproduces with probability
        self.maxBirthProb * (1 - popDensity).

        If this virus particle reproduces, then reproduce() creates and returns
        the instance of the offspring SimpleVirus (which has the same
        maxBirthProb and clearProb values as its parent).

        popDensity: the population density (a float), defined as the current
        virus population divided by the maximum population.

        returns: a new instance of the SimpleVirus class representing the
        offspring of this virus particle. The child should have the same
        maxBirthProb and clearProb values as this virus. Raises a
        NoChildException if this virus particle does not reproduce.
        """

        self.popDensity = popDensity
        proba = random.random()
        if proba <= self.maxBirthProb * (1 - self.popDensity):
            return SimpleVirus(self.maxBirthProb, self.clearProb)
        else:
            raise NoChildException
```

IPython console

Console 1/A

In [16]:

Permissions: RW | End-of-lines: CRLF | Encoding: ASCII | Line: 571 | Column: 1 | Memory: 88 %

Patient class

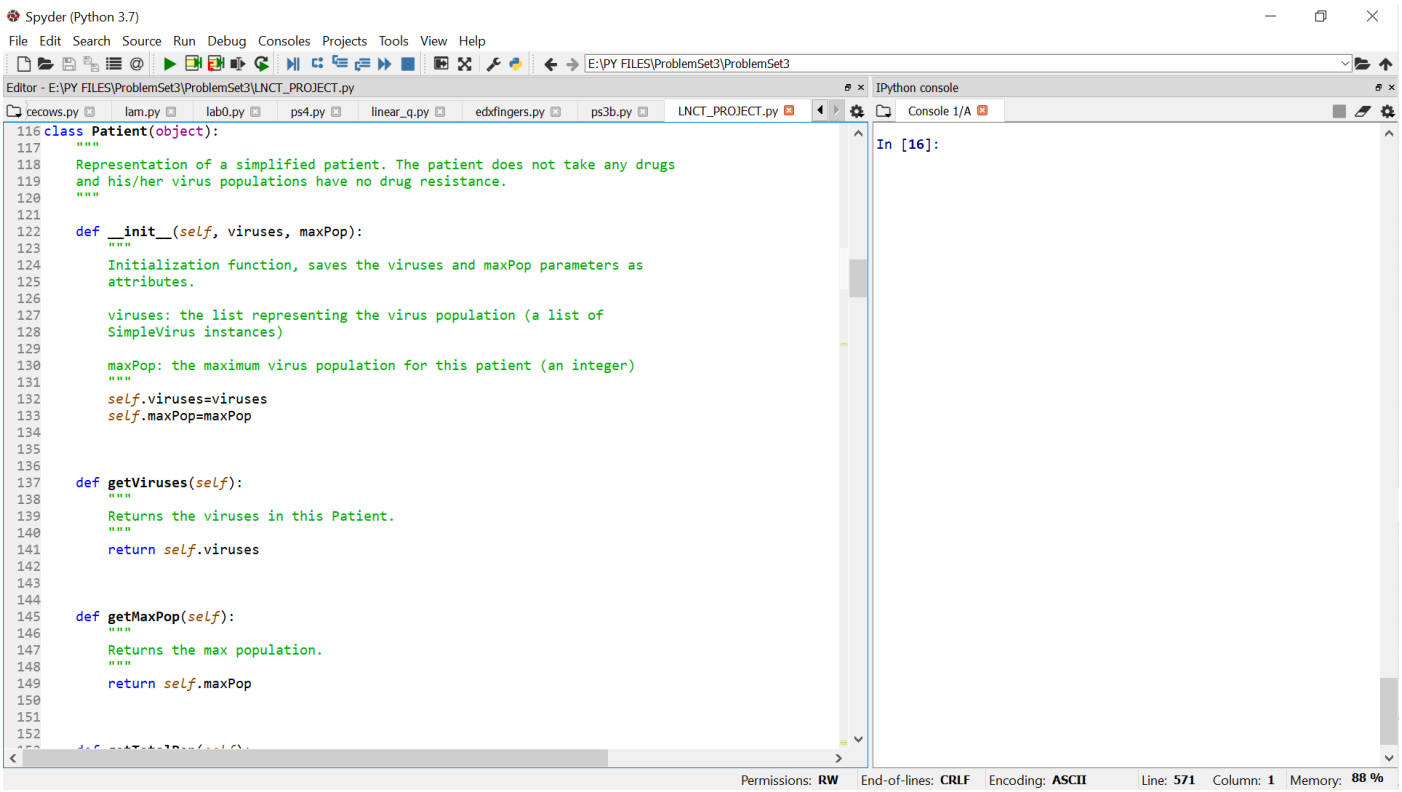
Implemented the `Patient` class, which maintains the state of a virus population associated with a patient.
The `update` method in the `Patient` class is the inner loop of the simulation. It modifies the state of the virus population for a single time step and returns the total virus population at the 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. Utilized the population density correctly which resulted in ending up at a situation at which I shouldn't need to provide an explicit check that the virus population exceeds `maxPop` when calculating how many offspring are added to the population -- just calculated the new population density and used that for the next call to `update`.

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.

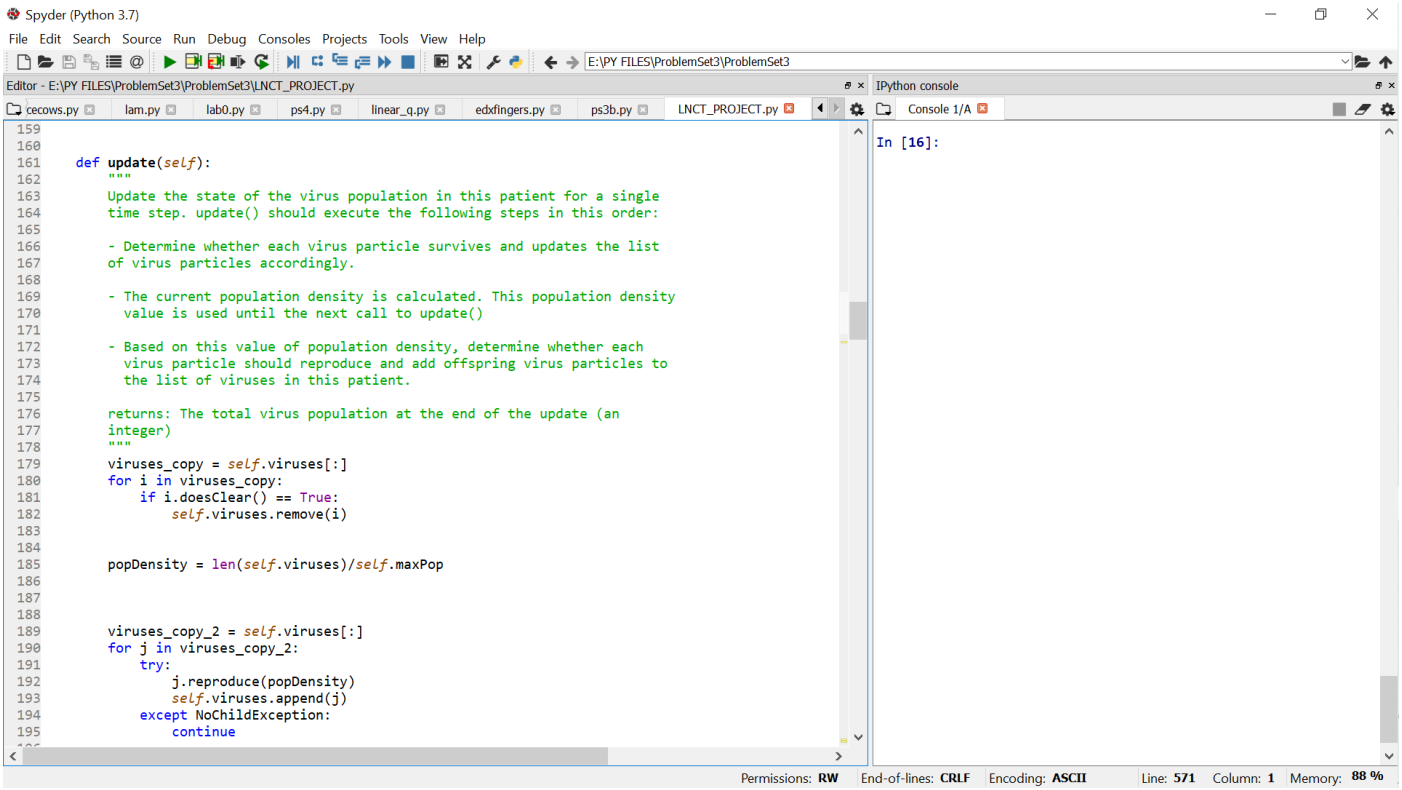
To summarize, `update` should first decide which virus particles are cleared and which survive by making use of the `doesClear` method of each `SimpleVirus` instance, then update the collection of `SimpleVirus` instances accordingly.

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.




```
class Patient(object):
    """
    Representation of a simplified patient. The patient does not take any drugs
    and his/her virus populations have no drug resistance.
    """

    def __init__(self, viruses, maxPop):
        """
        Initialization function, saves the viruses and maxPop parameters as
        attributes.

        viruses: the list representing the virus population (a list of
        SimpleVirus instances)

        maxPop: the maximum virus population for this patient (an integer)
        """
        self.viruses=viruses
        self.maxPop=maxPop



    def getViruses(self):
        """
        Returns the viruses in this Patient.
        """
        return self.viruses



    def getMaxPop(self):
        """
        Returns the max population.
        """
        return self.maxPop
```




```
    def update(self):
        """
        Update the state of the virus population in this patient for a single
        time step. update() should execute the following steps in this order:

        - Determine whether each virus particle survives and updates the list
        of virus particles accordingly.

        - The current population density is calculated. This population density
          value is used until the next call to update()

        - Based on this value of population density, determine whether each
          virus particle should reproduce and add offspring virus particles to
          the list of viruses in this patient.

        returns: The total virus population at the end of the update (an
        integer)
        """
        viruses_copy = self.viruses[:]
        for i in viruses_copy:
            if i.doesClear() == True:
                self.viruses.remove(i)


        popDensity = len(self.viruses)/self.maxPop



        viruses_copy_2 = self.viruses[:]
        for j in viruses_copy_2:
            try:
                j.reproduce(popDensity)
                self.viruses.append(j)
            except NoChildException:
                continue
```

Running and Analyzing a Simple Simulation (No Drug Treatment)

`simulationWithoutDrug(numViruses, maxPop, maxBirthProb, clearProb, numTrials)` that instantiates a `Patient`, simulates changes to the virus population for 300 time steps (i.e., 300 calls to `update`), and plots the average size of the virus population as a function of time; that is, the x-axis should correspond to the number of elapsed time steps, and the y-axis should correspond to the average size of the virus population in the patient. The population at time=0 is the population after the first call to `update`.
Run the simulation for `numTrials` trials, where `numTrials` in this case can be up to 100 trials. Use pylab to produce a plot (with a single curve) that displays the average result of running the simulation for many trials.

Called `simulationWithoutDrug` with the following parameters:

- `numViruses` = 100
- `maxPop` (maximum sustainable virus population) = 1000
- `maxBirthProb` (maximum reproduction probability for a virus particle) = 0.1
- `clearProb` (maximum clearance probability for a virus particle) = 0.05

Thus, your simulation should be instantiatating one `Patient` with a list of 100 `SimpleVirus` instances.
Each `SimpleVirus` instance in the viruses list should be initialized with the proper values for `maxBirthProb` and `clearProb`.

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

E:\PY FILES\ProblemSet3\ProblemSet3

Editor - E:\PY FILES\ProblemSet3\ProblemSet3\LNCT_PROJECT.py

ecows.py | lam.py | lab0.py | ps4.py | linear_q.py | edxfingers.py | ps3b.py | LNCT_PROJECT.py*

```python
def simulationWithoutDrug(numViruses, maxPop, maxBirthProb, clearProb,numTrials):
    """
    Run the simulation and plot the graph for problem 3 (no drugs are used,
    viruses do not have any drug resistance).
    For each of numTrials trial, instantiates a patient, runs a simulation
    for 300 timesteps, and plots the average virus population size as a
    function of time.

    numViruses: number of SimpleVirus to create for patient (an integer)
    maxPop: maximum virus population for patient (an integer)
    maxBirthProb: Maximum reproduction probability (a float between 0-1)
    clearProb: Maximum clearance probability (a float between 0-1)
    numTrials: number of simulation runs to execute (an integer)

    """


    viruses=[SimpleVirus(maxBirthProb,clearProb) for i in range(numViruses)]

    list_step=[0 for i in range(300)]

    for trial in range(numTrials):
        viruses_copy=viruses[:]

        a=Patient(viruses_copy,maxPop)

        for step in range(300):
            a.update()
            list_step[step]=list_step[step]+a.getTotalPop()

    averages=[pop/numTrials for pop in list_step ]

    pylab.plot(averages, label = "SimpleVirus")
    pylab.title("SimpleVirus simulation")
    pylab.xlabel("Time Steps")
    pylab.ylabel("Average Virus Population")
    pylab.legend(loc = "best")
```

IPython console

Console 1/A

```
In [16]:
```

Permissions: RW | End-of-lines: CRLF | Encoding: ASCII | Line: 204 | Column: 71 | Memory: 82 %

Results obtained :

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

E:\PY FILES\ProblemSet3\ProblemSet3

Editor - E:\PY FILES\ProblemSet3\ProblemSet3\LNCT_PROJECT.py

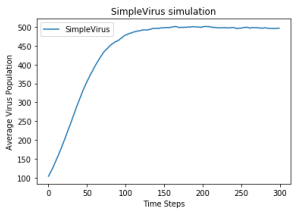
ecows.py | lam.py | lab0.py | ps4.py | linear_q.py | edxfingers.py | ps3b.py | LNCT_PROJECT.py

```python
        patient = TreatedPatient(virus_copy, maxPop)

        for step in range(150):
            patient.update()
            list_step_without_prescription[step]=list_step_without_prescription[step]+patient.getTot
            list_step_with_prescription[step]=list_step_with_prescription[step]+patient.getResistPop

        patient.addPrescription('guttagonol')

        for step in range(150,300):
            patient.update()
            list_step_without_prescription[step]=list_step_without_prescription[step]+patient.getTot
            list_step_with_prescription[step]=list_step_with_prescription[step]+patient.getResistPop


    average_without_prescription=[pop/numTrials for pop in list_step_without_prescription]
    average_with_prescription=[pop/numTrials for pop in list_step_with_prescription]
    pylab.plot(average_without_prescription,'r-',label="Without Prescription Trend")
    pylab.plot(average_with_prescription,'g-',label="With Prescription Trend")

    pylab.title(" Virus Simulation")
    pylab.xlabel("Time Steps")
    pylab.ylabel("Average Non-Resistant Virus Population and Resistant Virus Population ")
    pylab.legend(loc = "best")
    pylab.show()



#doing my test
#simulationWithoutDrug(numViruses, maxPop, maxBirthProb, clearProb, numTrials)
#simulationWithDrug(numViruses, maxPop, maxBirthProb, clearProb, resistances,mutProb, numTrials)
simulationWithoutDrug(100, 1000, 0.1,0.05,200)
#simulationWithDrug(100, 1000, 0.1, 0.05, {'guttagonol': False}, 0.005, 100)
```

IPython console

Console 1/A

```
In [15]: runfile('E:/PY FILES/ProblemSet3/ProblemSet3/
LNCT_PROJECT.py', wdir='E:/PY FILES/ProblemSet3/ProblemSet3')
```




```
In [16]:
```

Permissions: RW | End-of-lines: CRLF | Encoding: ASCII | Line: 571 | Column: 1 | Memory: 85 %

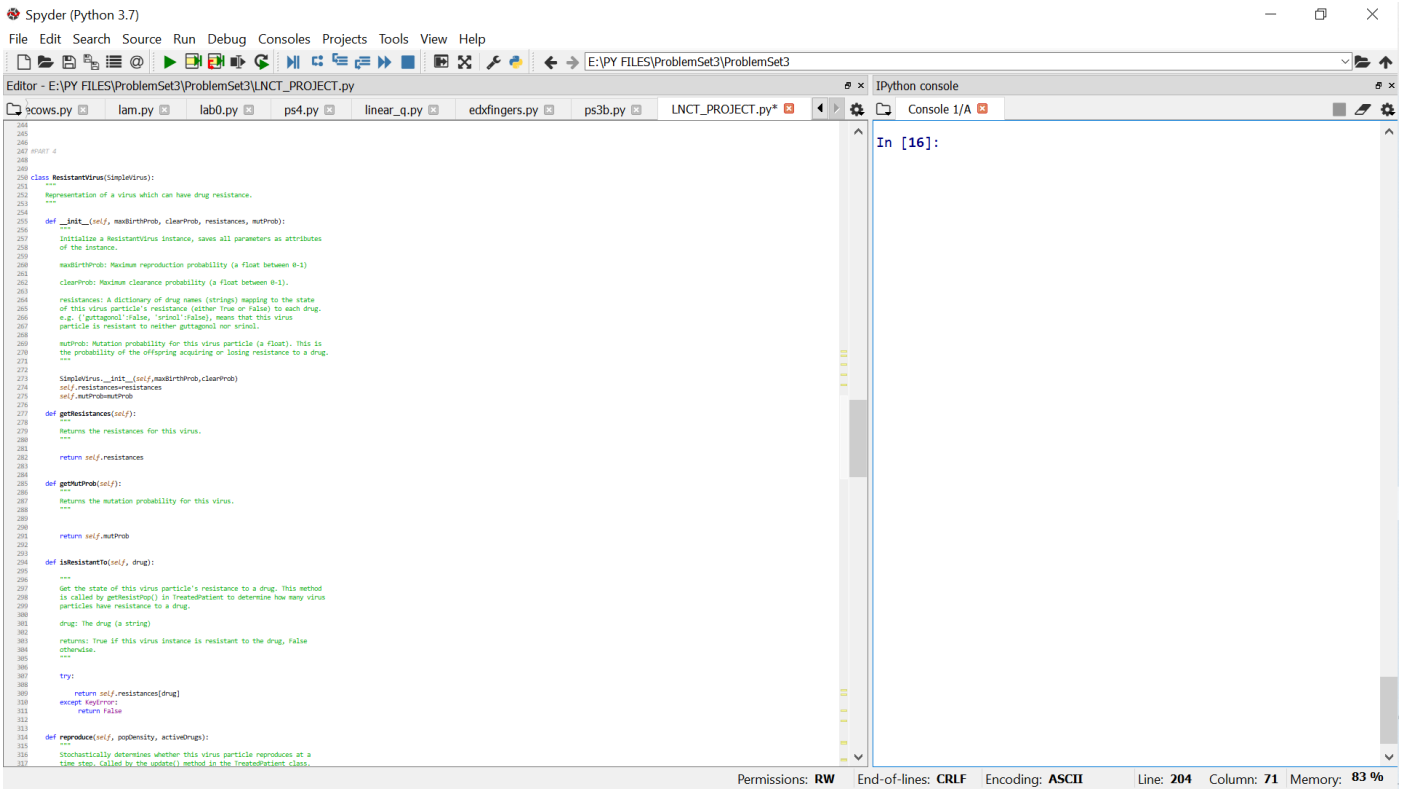
Implementing a Simulation With Drugs

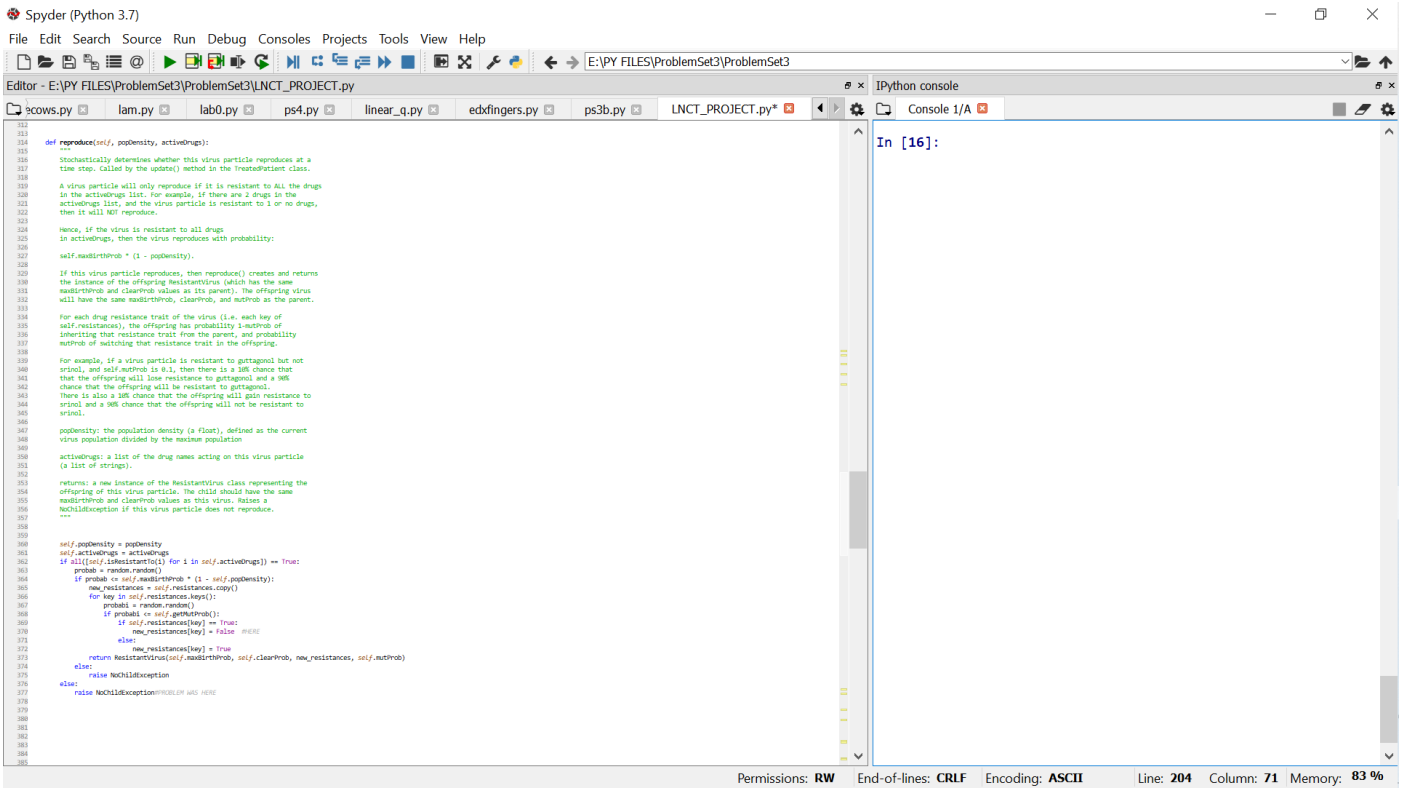
In this problem, we consider the effects of both administering drugs to the patient and the ability of virus particle offsprings to inherit or mutate genetic traits that confer drug resistance. As the virus population reproduces, mutations will occur in the virus offspring, adding genetic diversity to the virus population. Some virus particles gain favorable mutations that confer resistance to drugs.

ResistantVirus class

In order to model this effect, we introduce a subclass of `SimpleVirus` called `ResistantVirus`. `ResistantVirus` maintains the state of a virus particle's drug resistances, and accounts for the inheritance of drug resistance traits to offspring. Implement the `ResistantVirus` class.

(zoom_in to see clearly)





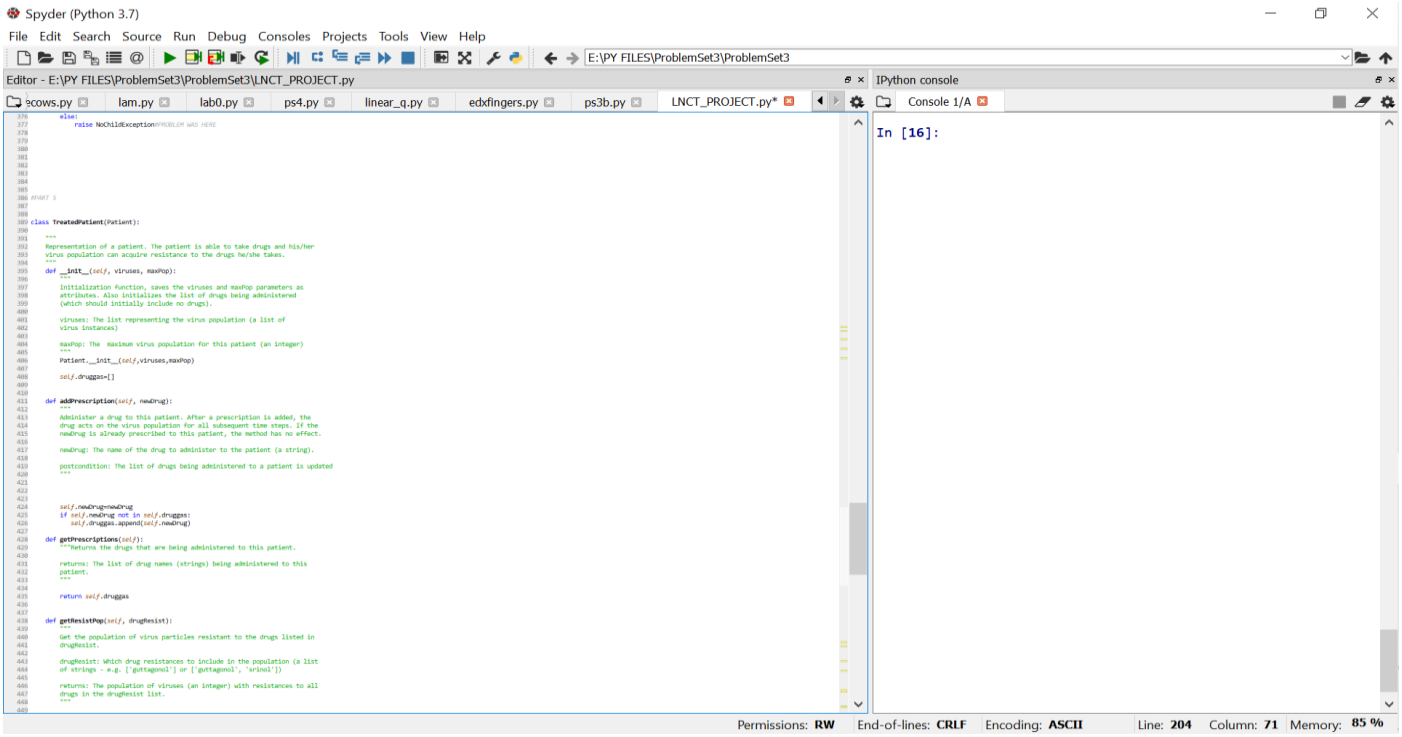

```
    def reproduce(self, popDensity, activeDrugs):
        """
        Stochastically determines whether this virus particle reproduces at a
        time step. Called by the update() method in the TreatedPatient class.

        A virus particle will only reproduce if it is resistant to ALL the drugs
        in the activeDrugs list. For example, if there are 2 drugs in the
        activeDrugs list, and the virus particle is resistant to 1 or no drugs,
        then it will NOT reproduce.

        Hence, if the virus is resistant to all drugs
        in activeDrugs, then the virus reproduces with probability:

        self.maxBirthProb * (1 - popDensity).

        If this virus particle reproduces, then reproduce() creates and returns
        the instance of the offspring ResistantVirus (which has the same
        maxBirthProb and clearProb values as its parent). The offspring virus
        will have the same maxBirthProb, clearProb, and mutProb as the parent.

        For each drug resistance trait of the virus (i.e. each key of
        self.resistances), the offspring has probability 1-mutProb of
        inheriting that resistance trait from the parent, and probability
        mutProb of switching that resistance trait in the offspring.

        For example, if a virus particle is resistant to guttagonol but not
        srinol, and self.mutProb is 0.1, then there is a 10% chance that
        that the offspring will lose resistance to guttagonol and a 90%
        chance that the offspring will be resistant to guttagonol.
        There is also a 10% chance that the offspring will gain resistance to
        srinol and a 90% chance that the offspring will not be resistant to
        srinol.

        popDensity: the population density (a float), defined as the current
        virus population divided by the maximum population

        activeDrugs: a list of the drug names acting on this virus particle
        (a list of strings).

        returns: a new instance of the ResistantVirus class representing the
        offspring of this virus particle. The child should have the same
        maxBirthProb and clearProb values as this virus. Raises a
        NoChildException if this virus particle does not reproduce.
        """


        self.popDensity = popDensity
        self.activeDrugs = activeDrugs
        if all([self.isResistantTo(i) for i in self.activeDrugs]) == True:
            probab = random.random()
            if probab <= self.maxBirthProb * (1 - self.popDensity):
                new_resistances = self.resistances.copy()
                for key in self.resistances.keys():
                    probabi = random.random()
                    if probabi <= self.getMutProb():
                        if self.resistances[key] == True:
                            new_resistances[key] = False  #HERE
                        else:
                            new_resistances[key] = True
                return ResistantVirus(self.maxBirthProb, self.clearProb, new_resistances, self.mutProb)
            else:
                raise NoChildException
        else:
            raise NoChildException#PROBLEM WAS HERE
```

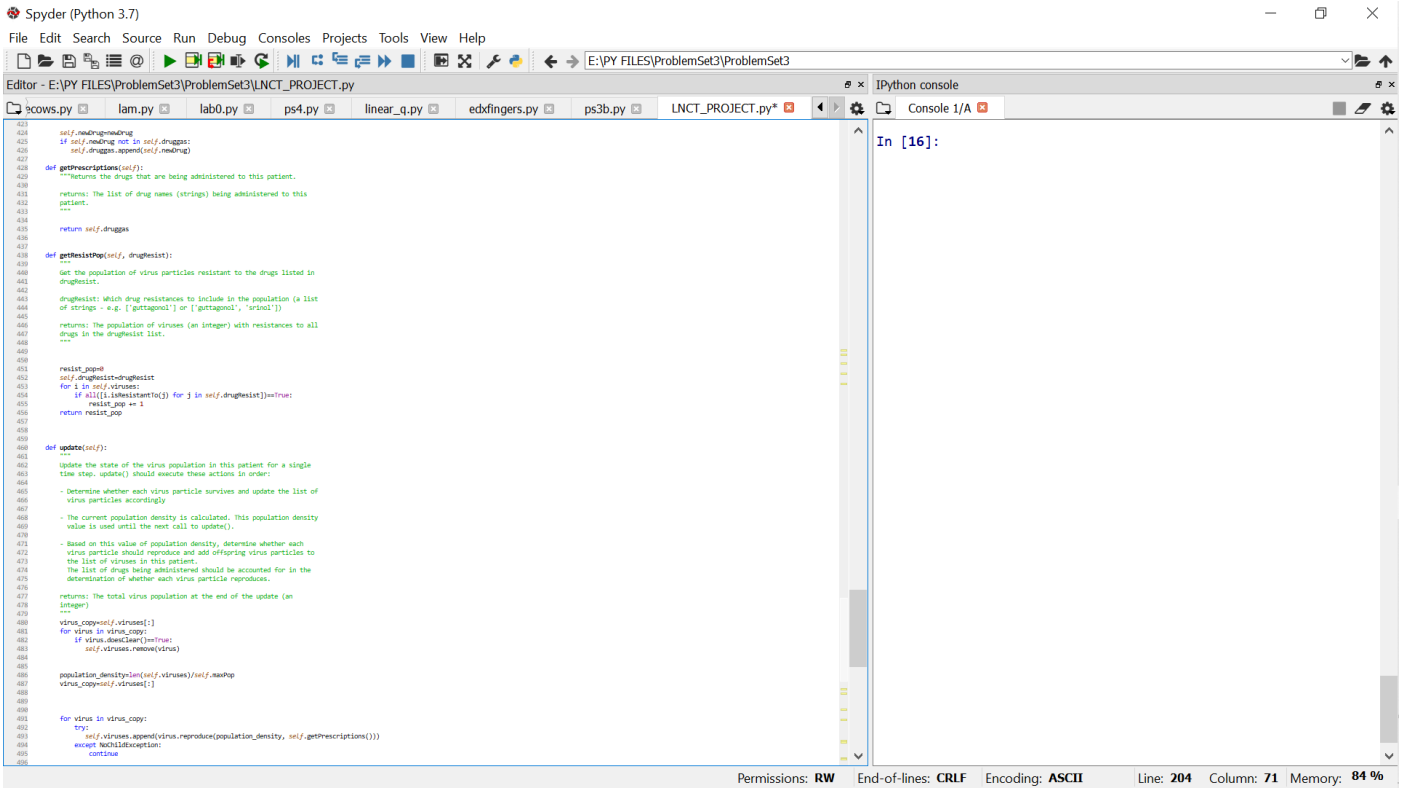
TreatedPatient Class

Needed a representation for a patient that accounts for the use of drug treatments and manages a collection of `ResistantVirus` instances. For that introduced the `TreatedPatient` class, which is a subclass of `Patient`. `TreatedPatient` must make use of the new methods in `ResistantVirus` and maintain the list of drugs that are administered to the patient.

Drugs are given to the patient using the TreatedPatient class's `addPrescription()` method. What happens when a drug is introduced? The drugs we consider do not directly kill virus particles lacking resistance to the drug, but prevent those virus particles from reproducing (much like actual drugs used to treat HIV). Virus particles with resistance to the drug continue to reproduce normally. Implement the `TreatedPatient` class.

(zoom_in_to_view)





Running and Analyzing a Simulation With a Drug

Used the implementation filled beforeto run a simulation. Created a `TreatedPatient` instance with the following parameters, then run the simulation:

- `viruses`, a list of 100 `ResistantVirus` instances

- `maxPop`, maximum sustainable virus population = 1000

Each `ResistantVirus` instance in the viruses list should be initialized with the following parameters:

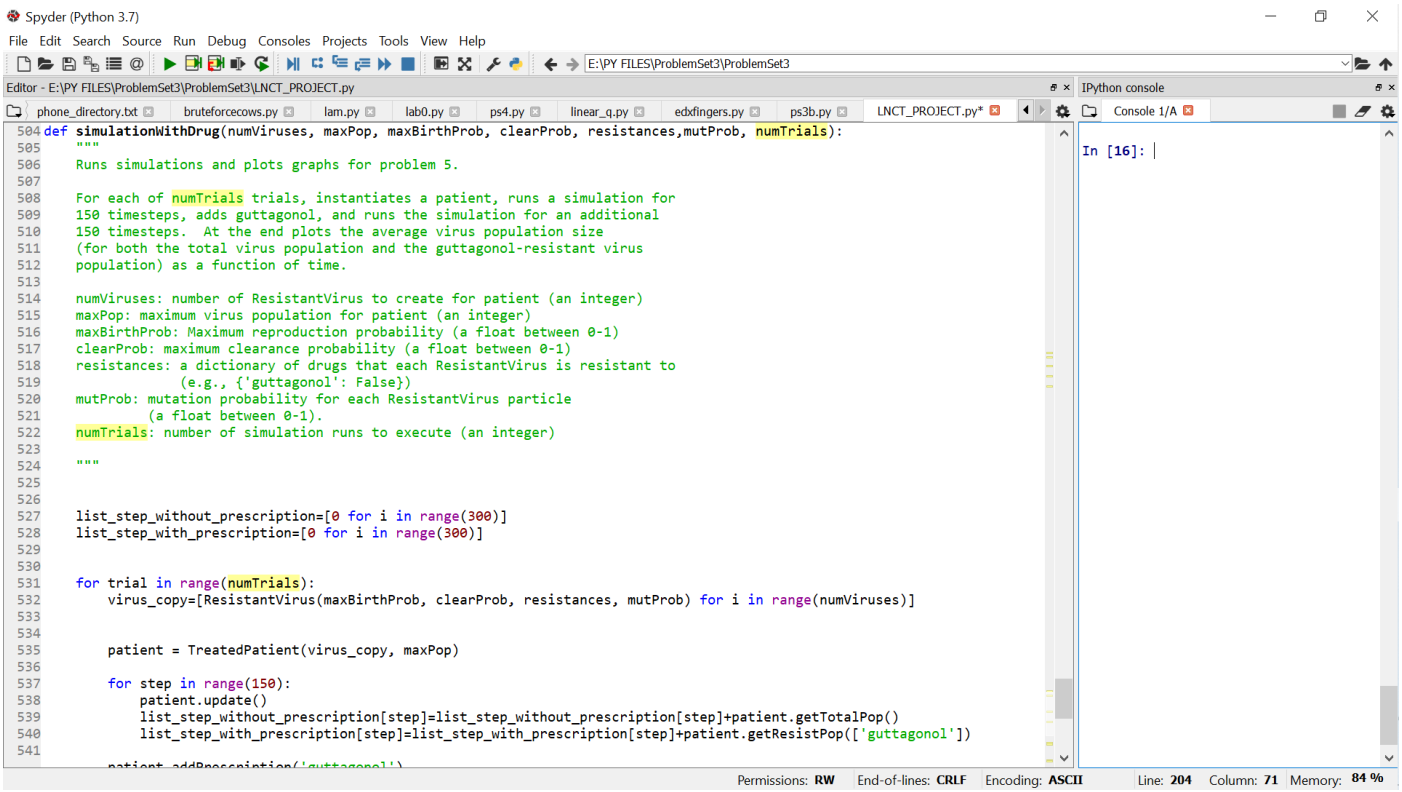
- `maxBirthProb`, maximum reproduction probability for a virus particle = 0.1
- `clearProb`, maximum clearance probability for a virus particle = 0.05
- `resistances`, The virus's genetic resistance to drugs in the experiment = {'guttagonol': False}
- `mutProb`, probability of a mutation in a virus particle's offspring = 0.005

Ran simulation that consists of 150 time steps, followed by the addition of the drug, guttagonol, followed by another 150 time steps.

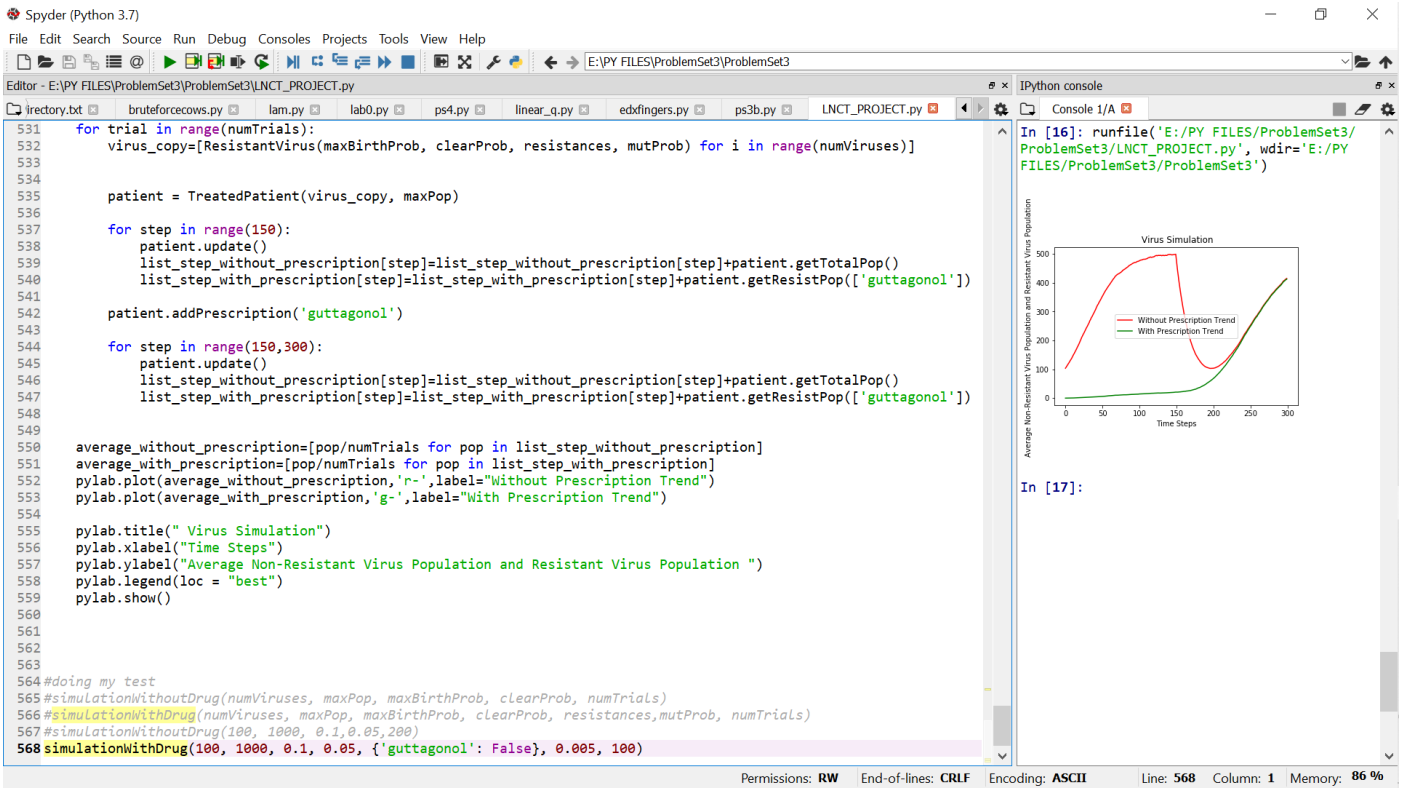
Used of the function `simulationWithDrug(numViruses, maxPop, maxBirthProb, clearProb, resistances, mutProb, numTrials` and performed up to 100 trials and made sure that results were repeatable and representative.

Created one plot that recorded both the average total virus population and the average population of guttagonol-resistant virus particles over time.

And considered some aspects like : What trends do you observe? Are the trends consistent with your intuition?




```
def simulationWithDrug(numViruses, maxPop, maxBirthProb, clearProb, resistances,mutProb, numTrials):
    """
    Runs simulations and plots graphs for problem 5.

    For each of numTrials trials, instantiates a patient, runs a simulation for
    150 timesteps, adds guttagonol, and runs the simulation for an additional
    150 timesteps.  At the end plots the average virus population size
    (for both the total virus population and the guttagonol-resistant virus
    population) as a function of time.

    numViruses: number of ResistantVirus to create for patient (an integer)
    maxPop: maximum virus population for patient (an integer)
    maxBirthProb: Maximum reproduction probability (a float between 0-1)
    clearProb: maximum clearance probability (a float between 0-1)
    resistances: a dictionary of drugs that each ResistantVirus is resistant to
                 (e.g., {'guttagonol': False})
    mutProb: mutation probability for each ResistantVirus particle
             (a float between 0-1).
    numTrials: number of simulation runs to execute (an integer)

    """


    list_step_without_prescription=[0 for i in range(300)]
    list_step_with_prescription=[0 for i in range(300)]


    for trial in range(numTrials):
        virus_copy=[ResistantVirus(maxBirthProb, clearProb, resistances, mutProb) for i in range(numViruses)]


        patient = TreatedPatient(virus_copy, maxPop)

        for step in range(150):
            patient.update()
            list_step_without_prescription[step]=list_step_without_prescription[step]+patient.getTotalPop()
            list_step_with_prescription[step]=list_step_with_prescription[step]+patient.getResistPop(['guttagonol'])

        patient.addPrescription('guttagonol')
```




```
    for trial in range(numTrials):
        virus_copy=[ResistantVirus(maxBirthProb, clearProb, resistances, mutProb) for i in range(numViruses)]


        patient = TreatedPatient(virus_copy, maxPop)

        for step in range(150):
            patient.update()
            list_step_without_prescription[step]=list_step_without_prescription[step]+patient.getTotalPop()
            list_step_with_prescription[step]=list_step_with_prescription[step]+patient.getResistPop(['guttagonol'])

        patient.addPrescription('guttagonol')

        for step in range(150,300):
            patient.update()
            list_step_without_prescription[step]=list_step_without_prescription[step]+patient.getTotalPop()
            list_step_with_prescription[step]=list_step_with_prescription[step]+patient.getResistPop(['guttagonol'])


    average_without_prescription=[pop/numTrials for pop in list_step_without_prescription]
    average_with_prescription=[pop/numTrials for pop in list_step_with_prescription]
    pylab.plot(average_without_prescription,'r-',label="Without Prescription Trend")
    pylab.plot(average_with_prescription,'g-',label="With Prescription Trend")

    pylab.title(" Virus Simulation")
    pylab.xlabel("Time Steps")
    pylab.ylabel("Average Non-Resistant Virus Population and Resistant Virus Population ")
    pylab.legend(loc = "best")
    pylab.show()



#doing my test
#simulationWithoutDrug(numViruses, maxPop, maxBirthProb, clearProb, numTrials)
#simulationWithDrug(numViruses, maxPop, maxBirthProb, clearProb, resistances,mutProb, numTrials)
#simulationWithoutDrug(100, 1000, 0.1,0.05,200)
simulationWithDrug(100, 1000, 0.1, 0.05, {'guttagonol': False}, 0.005, 100)
```

CODES: https://drive.google.com/drive/folders/1PEbsRDEMV2yJLBkxthiUHInhmrh85bXJ?usp=sharing (.txt and .py files)

Bibliography And Resources :

REFERENCES - 1. Introduction to programming and data science with python by John V Guttag &
. 2. Edx courses 6.00.2x (especially pset 3) and 6.00.1x

Certification (For authenticity of the work) - https://www.linkedin.com/in/prince-dwivedi
OR
. https://courses.edx.org/certificates/2a92171aa82a436ea2c7a45691115567