SFWRTECH 3PR3:

Procedural and Objective Oriented Programming Concepts

(Assignment #3)

Student Name: Dojae Kim

Student Number: 400420323

Professor Name: Dr. Seshasai Srinivasan

**Objective**

The purpose of this Assignment 3:

1. To give students understanding python structure.

2. To write, test, and debug python programs.

3. To implement, and execute conditional statement iteratively until a given condition is satisfied.

4. To understand how to import various modules from python library.

**Equation**

**Part A:**

1. *V = u + gt*

2. *d = ut + 0.5gt2*

**Part B:**

1. *E\_potential = mgy*

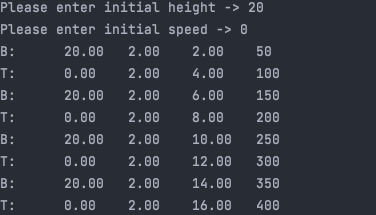
2. *E\_kinetic = 0.5mv2*

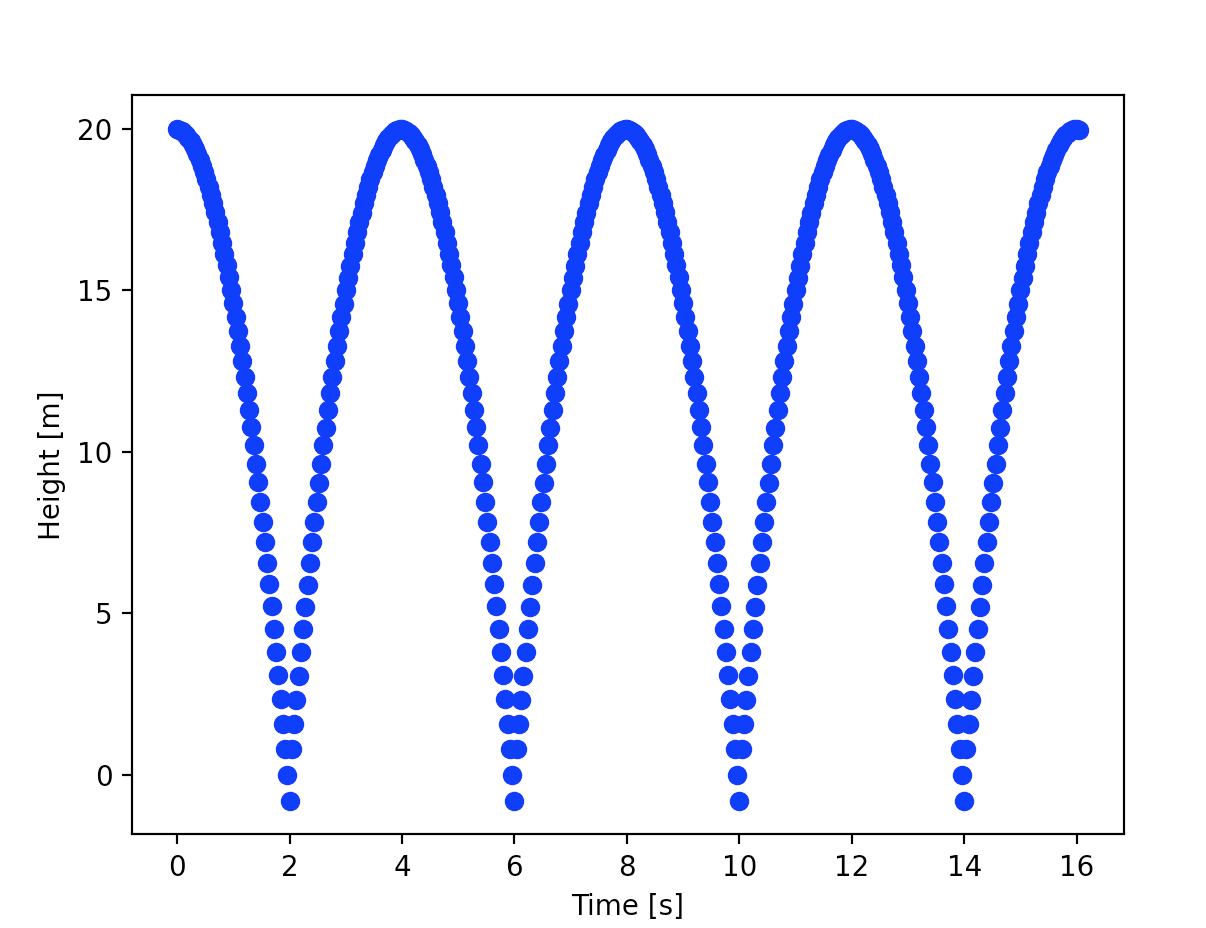
**Part A** **Source Code**

# Student name: Dojae Kim  
# Student number: 400420323  
# Student email: kim408@mcmaster.ca  
# Lecture: SFWRTECH 3PR3  
# Assignment 3 Part A  
  
from typing import Tuple  
import matplotlib.pyplot as plt  
  
  
def getInput() -> Tuple[float, float]:  
 """  
 Get input from user - tuple of 2 floats, 1st one is initial height and  
 2nd one is initial speed  
 :return: Tuple of initial values  
 """  
 done = False  
 init\_height = 0  
 init\_speed = 0  
 while not done:  
 try:  
 init\_height = float(input("Please enter initial height -> "))  
 if init\_height >= 0:  
 done = True  
 except ValueError:  
 print("Initial height should be positive!")  
  
 done = False  
 while not done:  
 try:  
 init\_speed = float(input("Please enter initial speed -> "))  
 if init\_speed >= 0:  
 done = True  
 except ValueError:  
 print("Initial speed should be positive!")  
  
 return init\_height, init\_speed  
  
  
def motionSimulator(init\_values: Tuple[float, float]) -> None:  
 """  
 Simulate motion of the ball  
 :param init\_values: Initial values (height and speed)  
 :return: None  
 """  
 down = True # Initially ball travels down  
 v = init\_values[1]  
 h = init\_values[0]  
 g = 10  
 plt.plot(0, h, 'bo')  
 plt.xlabel('Time [s]')  
 plt.ylabel('Height [m]')  
 prev\_min\_max = 0  
 for i in range(0, 401):  
 time = (i + 1) \* 0.04  
 if down:  
 v += g \* 0.04  
 h -= (v \* 0.04) + (0.5 \* g \* 0.04 \*\* 2)  
 else:  
 v -= g \* 0.04  
 h += (v \* 0.04) + (0.5 \* g \* 0.04 \*\* 2)  
  
 plt.plot(time, h, 'bo')  
  
 if h < 0:  
 down = False  
 h = 0  
 print(format("B:\t%.2f\t%.2f\t%.2f\t%d") %  
 (v, time - prev\_min\_max, time, i + 1))  
 prev\_min\_max = time  
 if v < 0:  
 down = True  
 v = 0  
 print(format("T:\t%.2f\t%.2f\t%.2f\t%d") %  
 (v, time - prev\_min\_max, time, i + 1))  
 prev\_min\_max = time  
  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 user\_input = getInput()  
 motionSimulator(user\_input)

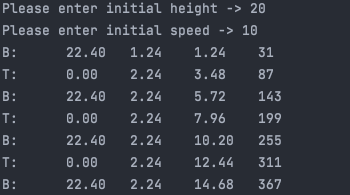
**Part A Sample Output**

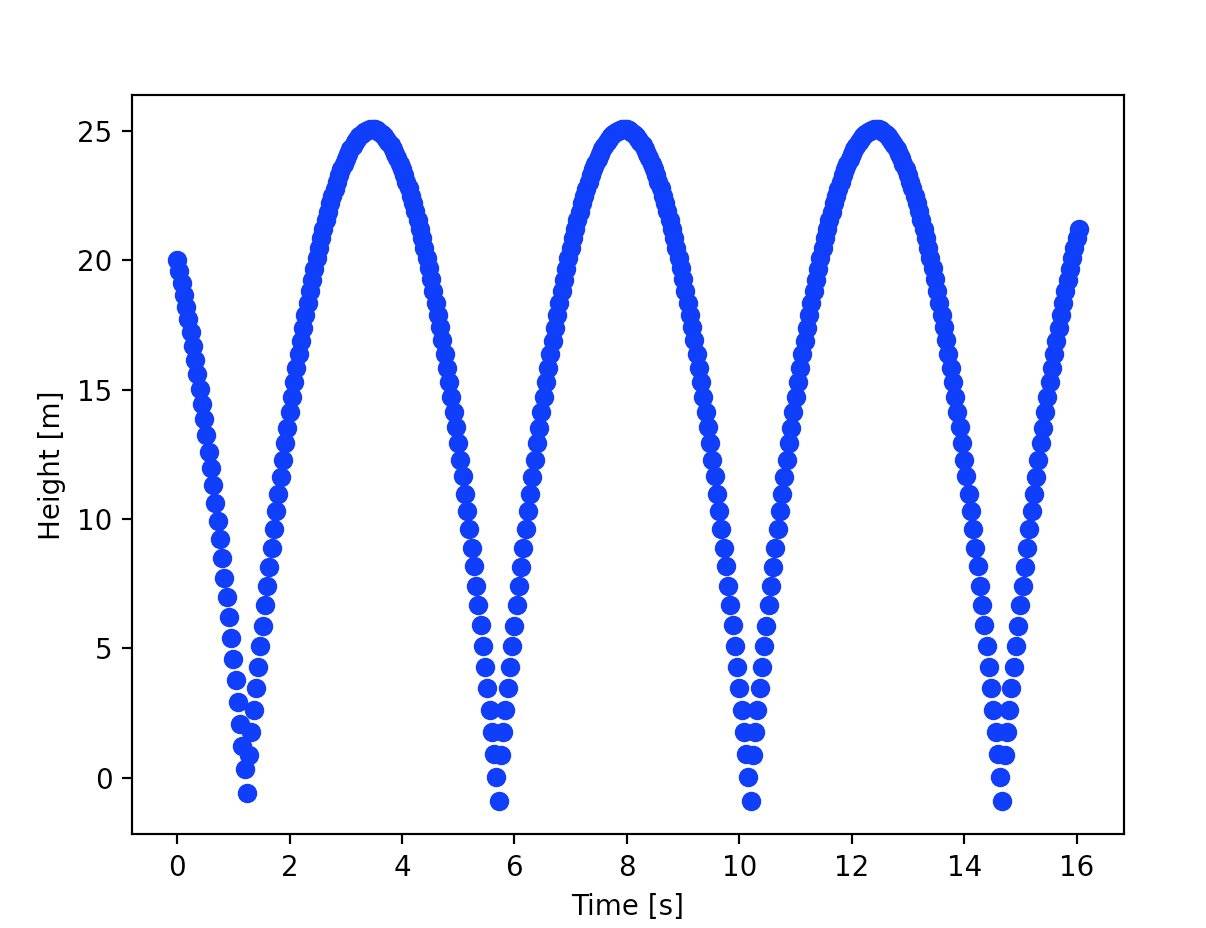
**Sample 1:**

****

****

**Sample 2:**



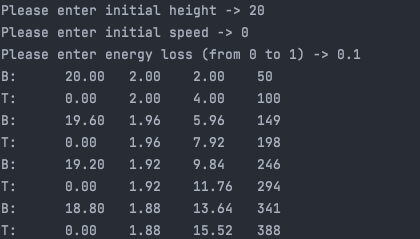


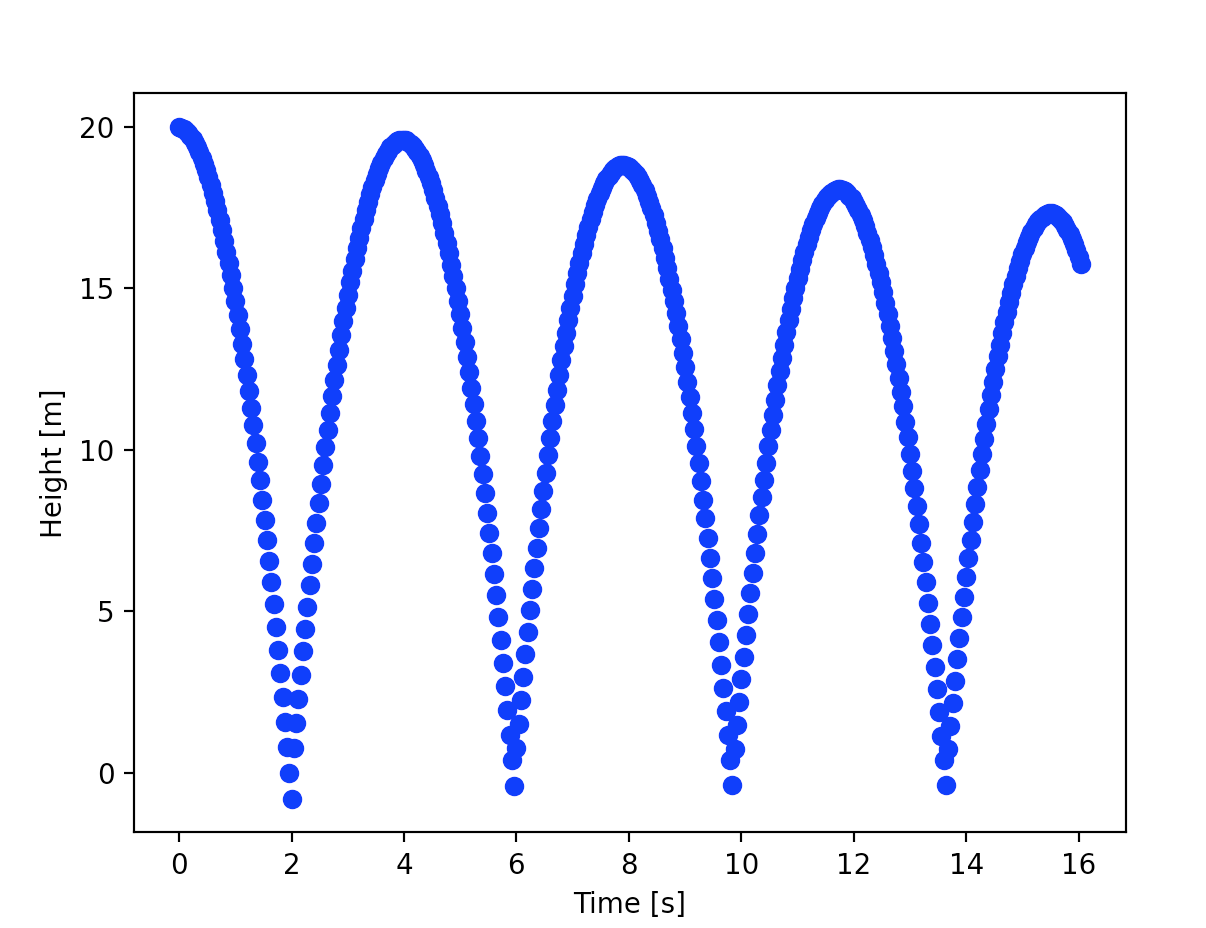
**Part B Source Code**

# Student name: Dojae Kim  
# Student number: 400420323  
# Student email: kim408@mcmaster.ca  
# Lecture: SFWRTECH 3PR3  
# Assignment 3 Part B  
  
from typing import Tuple  
import matplotlib.pyplot as plt  
  
  
def getInput() -> Tuple[float, float, float]:  
 """  
 Get input from user - tuple of 2 floats, 1st one is initial height,  
 2nd one is initial speed and 3rd one is energy loss (>= 0 and <=1)  
 :return: Tuple of initial values  
 """  
 done = False  
 init\_height = 0.0  
 init\_speed = 0.0  
 ep = 0.0  
 while not done:  
 try:  
 init\_height = float(input("Please enter initial height -> "))  
 if init\_height >= 0:  
 done = True  
 except ValueError:  
 print("Initial height should be positive!")  
  
 done = False  
 while not done:  
 try:  
 init\_speed = float(input("Please enter initial speed -> "))  
 if init\_speed >= 0:  
 done = True  
 except ValueError:  
 print("Initial speed should be positive!")  
  
 done = False  
 while not done:  
 try:  
 ep = float(input("Please enter energy loss (from 0 to 1) -> "))  
 if 0 <= ep <= 1:  
 done = True  
 else:  
 print("Energy loss should be between 0 and 1")  
 except ValueError:  
 print("Energy loss should be positive!")  
  
 return init\_height, init\_speed, ep  
  
  
def energyLoss(v: float, ep: float) -> float:  
 """  
 Calculate energy loss  
 :param v: Current speed  
 :param ep: Energy loss  
 :return: New speed considering energy loss  
 """  
 return v \* (1 - ep \*\* 2)  
  
  
def motionSimulator(init\_values: Tuple[float, float, float]) -> None:  
 """  
 Simulate motion of the ball  
 :param init\_values: Initial values (height and speed)  
 :return: None  
 """  
 down = True # Initially ball travels down  
 v = init\_values[1]  
 h = init\_values[0]  
 g = 10  
 plt.plot(0, h, 'bo')  
 plt.xlabel('Time [s]')  
 plt.ylabel('Height [m]')  
 prev\_min\_max = 0  
 for i in range(0, 401):  
 time = (i + 1) \* 0.04  
 if down:  
 v += g \* 0.04  
 h -= (v \* 0.04) + (0.5 \* g \* 0.04 \*\* 2)  
 else:  
 v -= g \* 0.04  
 h += (v \* 0.04) + (0.5 \* g \* 0.04 \*\* 2)  
  
 plt.plot(time, h, 'bo')  
  
 if h < 0:  
 down = False  
 h = 0  
 print(format("B:\t%.2f\t%.2f\t%.2f\t%d") %  
 (v, time - prev\_min\_max, time, i + 1))  
 prev\_min\_max = time  
 v = energyLoss(v, init\_values[2])  
 if v < 0:  
 down = True  
 v = 0  
 print(format("T:\t%.2f\t%.2f\t%.2f\t%d") %  
 (v, time - prev\_min\_max, time, i + 1))  
 prev\_min\_max = time  
  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 user\_input = getInput()  
 motionSimulator(user\_input)

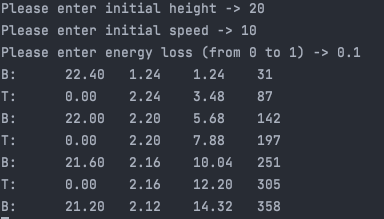
**Part B Sample Output**

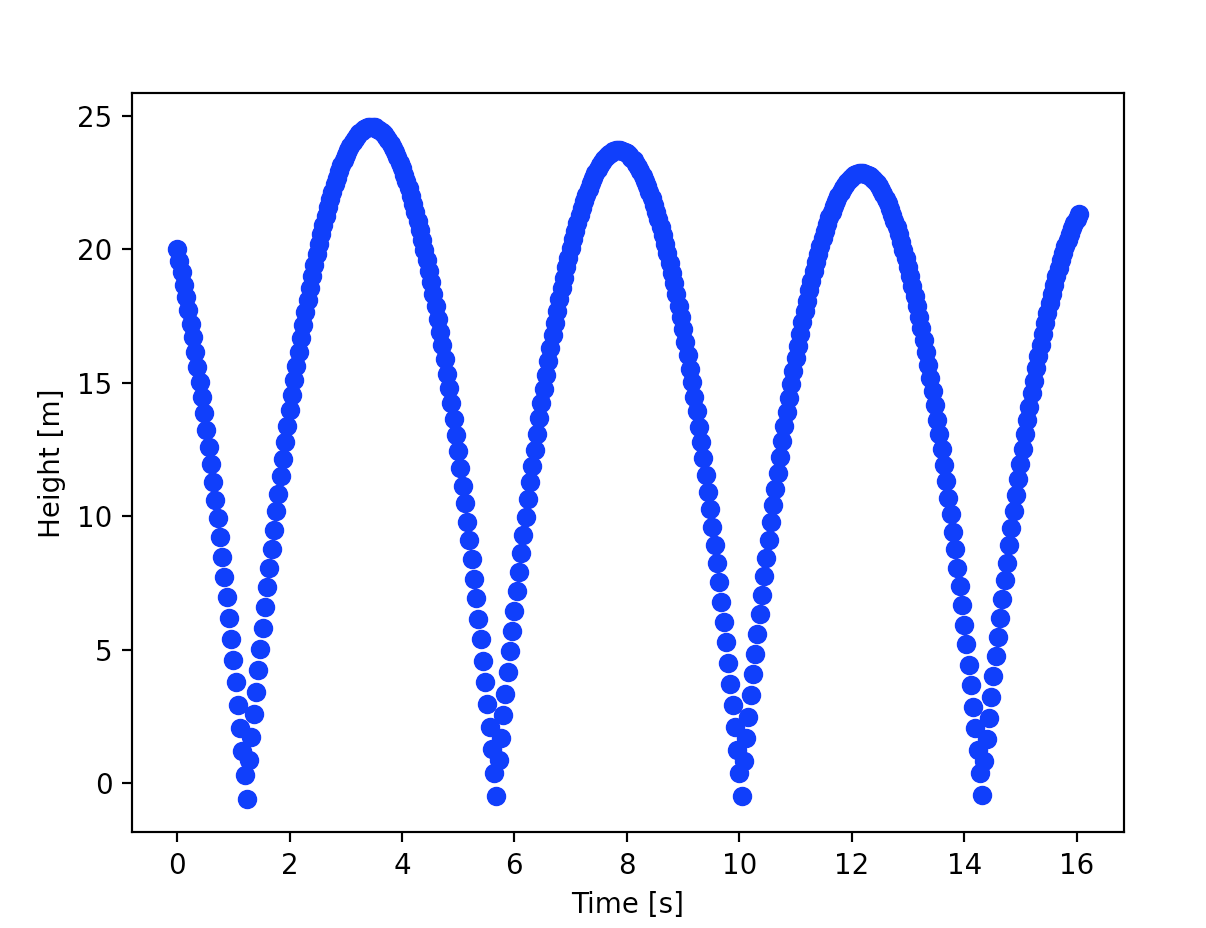
**Sample 1:**





**Sample 2:**





**Sample 3:**

