Plotting the Kinematic Equations – Creating a Basic Motion Simulator

This program is designed to reflect a real world example of behaviors of an object that either is free falling or launched upward or custom perform (position & velocity), with constant acceleration as earth's gravity.

The program launches with a "Welcome" message in the terminal, and asks the user for desired measurement system which are empirical or metric.

```
Welcome To The Motion Simulation!
Please Type Your Unit System(Empirical/Metric)
# Program welcomes the user
print("Welcome To The Motion Simulation!")
# Program then asks for unit system
print("Please Type Your Unit System(Empirical/Metric): ")
user_unit = str(input())
```

The program follows with asking for a mode selection in given options.

Once the user enters their choice, there will be another statement/s with the selected unit names in display (units are assigned by user selection for unit system) to define how high the object is at or/and what velocity object has. At last, user will be determining the time intervals as they wish.

```
Welcome To The Motion Simulation!

Please Type Your Unit System(Empirical/Metric):

metric

Please Choose Your Simulation Mode: Free Fall; Launch Upward; Custom

free fall

Please Enter the Vertical Distance of the Object: meters

100

Type Three Time Intervals To Simulate: seconds

Time @
```

In the example, user entered three time intervals as shown in the image below.

Then, the program output three statements.

As you can see, we get the velocity and the position of the object at the current time/s. In the output, positive or negative values for velocity determines the object rises or falls.

However, the third statement is printed different.

This is simply because of the object does not have a movement at given time, it stopped. Thus, the program prints out the message with the time and the velocity when the object hit the ground.

```
Type Three Time Intervals To Simulate: seconds

Time @ 1

Time @ 3

Time @ 10

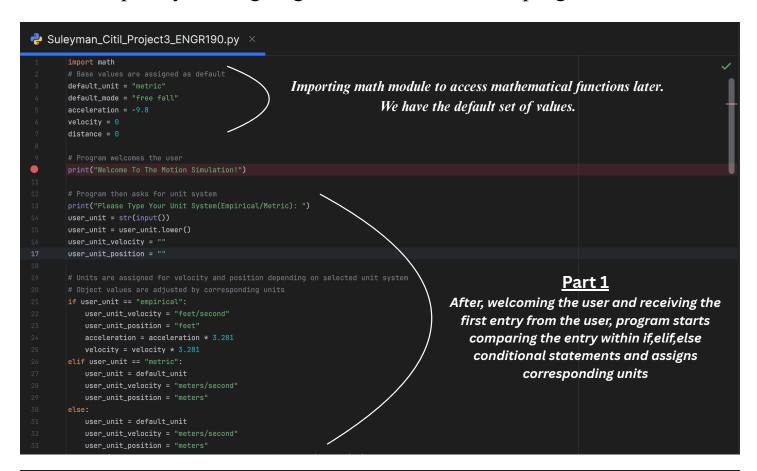
At 1.0 seconds, the velocity is -9.80 meters/second, the position is 95.10 meters.

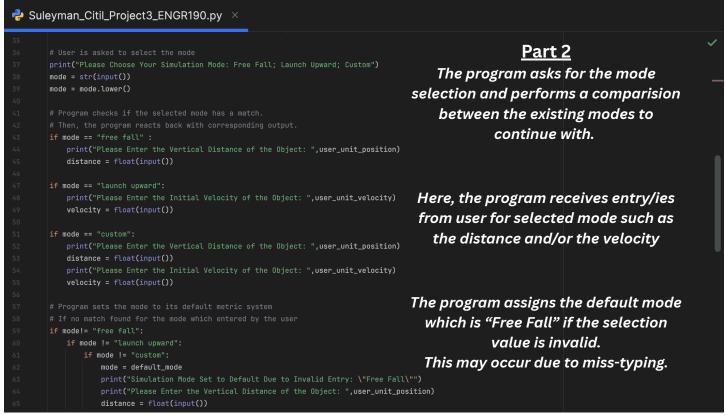
At 3.0 seconds, the velocity is -29.40 meters/second, the position is 55.90 meters.

Warning! Object has hit the ground. @,4.52 seconds with velocity of -44.27 meters/second.

Process finished with exit code 0
```

In this part, you are going to learn that how this program functions.





```
Suleyman_Citil_Project3_ENGR190.py ×

# The user enters three time intervals in seconds

# Then program assigns calculated positions and velocities to variables based on the given times

print("Type Three Time Intervals To Simulate: seconds")

time = [input("Time @ "), input("Time @ "), input("Time @ ")]

time_1 = float(time[0])

time_2 = float(time[1])

time_3 = float(time[2])

Velocity_1 = velocity + acceleration * time_1

distance_1 = distance + (velocity * time_1) + (acceleration * (time_1**2) / 2)

velocity_2 = velocity + acceleration * time_2

distance_2 = distance + (velocity * time_2) + (acceleration * (time_2***2) / 2)

velocity_3 = velocity + acceleration * time_3

distance_3 = distance + (velocity * time_3) + (acceleration * (time_3***2) / 2)
```

```
Suleyman_Citil_Project3_ENGR190.py ×
       Equation is 0.5 \times acceleration \times time.square + velocity \times time + height = 0
       (\Delta>0) and a maximum of two roots) or (\Delta=0) and a root) or (\Delta<0) and no roots no collision)
       Now, user can be warned for what the time and what the velocity are at the collision.
       a = acceleration * 0.5
                                                                                                 Part 3 continues
       b = velocity
       c = distance
                                                                      The program performs a critical calculation to determine when
       discriminant = pow(b.2) - (4 * a * c)
                                                                   the object reaches the ground and what its velocity at the moment.
       if discriminant > 0:
          root1 = (-math.sgrt(discriminant)-b)/2/a
          root2 = (math.sgrt(discriminant)-b)/2/a
                                                                               Equation is solved for time by finding the roots.
           collision_velocity = velocity + acceleration * collision_time
                                                                      This section helps to avoid logical errors that can be caused by
          collision_time = (-math.sqrt(discriminant)-b)/2/a
           collision_velocity = velocity + acceleration * collision_time
                                                                        printing greater negative values of velocity after the object
                                                                                              reaches/hits the ground.
           collision_time = None
           collision_velocity = None
```

```
"""Lastly the program outputs as comparing the distance"""

if distance_1 >= 0:
    print(f"At {time_1} seconds, the velocity is {velocity_1:.2f} {user_unit_velocity}, the position is {distance_1:.2f} {user_unit_position}.")

elif distance_1 < 0:
    print(f"Warning! Object has hit the ground. @,{collision_time:.2f} seconds with velocity of {collision_velocity:.2f} {user_unit_velocity}.")

if distance_2 >= 0:
    print(f"At {time_2} seconds, the velocity is {velocity_2:.2f} {user_unit_velocity}, the position is {distance_2:.2f} {user_unit_position}.")

elif distance_2 < 0:
    print(f"Warning! Object has hit the ground. @,{collision_time:.2f} seconds with velocity of {collision_velocity:.2f} {user_unit_velocity}.")

if distance_3 >= 0:
    print(f"At {time_3} seconds, the velocity is {velocity_3:.2f} {user_unit_velocity}, the position is {distance_3:.2f} {user_unit_position}.")

elif distance_3 < 0:
    print(f"Warning! Object has hit the ground. @,{collision_time:.2f} seconds with velocity of {collision_velocity:.2f} {user_unit_velocity}.")
```

Last image contains the conditional statements for what is printed out depending on the distance of the object for each moment. This conditionals allow the program to output a meaningful message instead of printing larger negative values in spite of the object is halted.

Example Of Real World Use In Engineering: S-P-D

- Safety: Before building a physical prototype, engineers can use this simulation to model different launch scenarios. By varying the initial velocity and acceleration (thrust), they can determine if the rocket will clear launch structures. This allows them to identify and mitigate potential hazards.
- **Performance**: The simulation can optimize launch performance. Engineers could test different initial thrust settings (velocity in this program) and launch heights (distance). This helps maximize payload and minimize operational costs.
- **Design**: The simulation informs design choices. For example, if the simulation shows that the rocket needs a higher initial velocity to avoid a certain obstacle, the engineers know they need to design more powerful initial-stage thrusters. Conversely, if the simulation shows the rocket reaching its target too early, they can design for less thrust, which might reduce structural stress and cost.