Week 1 & Week 2: The Double Pendulum

Accelerate

Alexander Van Doren [1]

1 Introduction

This codebase implements a real-time double pendulum simulation delivered as a small web application. The backend computes the physics continuously and exposes the current bob positions over a lightweight JSON endpoint, while the frontend renders the system in an HTML5 canvas and animates it in the browser.

1.1 Getting Started

You can find the skeleton code here.
You should **fork** the repository, and then run the command:

```
git clone https://github.com/your_github_username/double_pendulum
```

Once the code is cloned onto your computer you can start developing, but remember to **always commit regularly to GitHub**! If you don't commit at a minimum rate of $\approx 1 commit/hour$ we may not be able to award you your prizes!

1.2 Tech Stack

- **Language:** Python *(3.13 - Use a* `Virtual Environment`*)*
- **Web Framework:** Flask *(routing, templating)*
- **Frontend:** HTML5, CSS, and JavaScript *(*`Canvas`*)*

Affiliation [1] Hack Club
Correspondence alexvd@hackclub.com
Version September 10, 2025

1.3 What is Flask?

Flask is a minimal, flexible Python web framework for building HTTP services and server-rendered pages. In this project, Flask is used to:

- Serve the main HTML template at `/`.
- Expose a JSON endpoint (`/coords`) that returns the current coordinates of the pendulum bobs — this is useful for debugging.
- Run the Application on a Server.

1.4 Why Python?

Python is well-suited for numbercrunching and simulations:

- **Ease of Use:** It has a very clear syntax, which is helpful when using lots of Math in your projects.
- **Useful Libraries:** The python ecosystem is unparalleled when it comes to library support.

1.5 What We Consider a "Shipped" Project

For this challenge we will consider projects shipped if they are deployed on the internet, and you can provide a playable url, which when clicked will open up your version of the simulation.

2 Double Pendulum Class

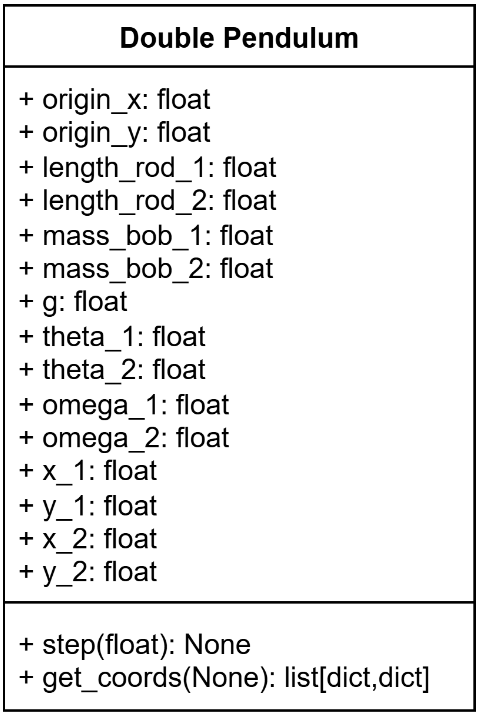

Figure 1: Caption

2.1 Mathematics behind the Modelling

Ideally you understand a little bit of this, but you don't need to understand all of it. Equations that are captioned as **important**, are, in fact, important.

2.1.1 Conventions and Symbols

We model a planar double pendulum with:

- Rod lengths: l_1, l_2; masses: m_1, m_2; gravity: g.
- Angles from the vertical (downward positive): θ_1, θ_2; angular velocities: ω_1, ω_2.
- The starting coordinates for each bob with y increasing downward are given by:

$$x_1 = x_0 + l_1 \sin\theta_1, \qquad y_1 = y_0 + l_1 \cos\theta_1,$$
$$x_2 = x_1 + l_2 \sin\theta_2, \qquad y_2 = y_1 + l_2 \cos\theta_2,$$

  where (x_0, y_0) is the fixed pivot.

2.1.2 Use of the Lagrangian (reference)

With angles measured from vertical, the kinetic and potential energies are

$$T = \tfrac{1}{2} m_1 l_1^2 \omega_1^2 + \tfrac{1}{2} m_2 \Big(l_1^2 \omega_1^2 + l_2^2 \omega_2^2 + 2 l_1 l_2 \omega_1 \omega_2 \cos(\theta_1 - \theta_2) \Big),$$
$$V = (m_1 + m_2) g l_1 (1 - \cos\theta_1) + m_2 g l_2 (1 - \cos\theta_2),$$

and $\mathcal{L} = T - V$. Applying Euler–Lagrange yields the coupled nonlinear ODEs below.

2.1.3 Equations of Motion (implemented form)

Let $\Delta = \theta_2 - \theta_1$ and define

$$D_1 = (m_1 + m_2)l_1 - m_2 l_1 \cos^2\Delta, \qquad D_2 = \frac{l_2}{l_1} D_1.$$

The angular accelerations used in the simulation are

$$\alpha_1 = \frac{m_2 l_1 \omega_1^2 \sin\Delta\cos\Delta + m_2 g \sin\theta_2 \cos\Delta + m_2 l_2 \omega_2^2 \sin\Delta - (m_1 + m_2) g \sin\theta_1}{D_1},$$

$$\alpha_2 = \frac{-m_2 l_2 \omega_2^2 \sin\Delta\cos\Delta + (m_1 + m_2) g \sin\theta_1 \cos\Delta - (m_1 + m_2) l_1 \omega_1^2 \sin\Delta - (m_1 + m_2) g \sin\theta_2}{D_2}.$$

When stepping through our simulation we use Eulers method, with a timestep of dt, which results in:

$$\omega_1 \mathrel{+}= \alpha_1\, dt, \quad \omega_2 \mathrel{+}= \alpha_2\, dt, \quad \theta_1 \mathrel{+}= \omega_1\, dt, \quad \theta_2 \mathrel{+}= \omega_2\, dt.$$

Figure 2: This is **Important**

2.1.4 Numerical Methods

- The system is chaotic, and stability is sensitive to the timestep: dt. Moderate values of dt (e.g. 0.01–0.06) offer a good balance of speed and fidelity.

2.2 Features of the Class

2.2.1 Attributes A list of the Class's attributes:

- `origin_x`: X-coordinate of the fixed pivot (pixels).
- `origin_y`: Y-coordinate of the fixed pivot (pixels, downwards positive).
- `length_rod_1`: Length of the first rod L_1 (pixels).
- `length_rod_2`: Length of the second rod L_2 (pixels).
- `mass_bob_1`: Mass of the first bob m_1 (arbitrary units).
- `mass_bob_2`: Mass of the second bob m_2 (arbitrary units).
- `g`: Gravitational acceleration (pixels/s^2 after scaling).
- `theta_1`: Angle of the first bob from vertical (radians).
- `theta_2`: Angle of the second bob from vertical (radians).
- `omega_1`: Angular velocity of the first bob (rad/s).
- `omega_2`: Angular velocity of the second bob (rad/s).
- `x_1`: Current X-position of the first bob (pixels).
- `y_1`: Current Y-position of the first bob (pixels).
- `x_2`: Current X-position of the second bob (pixels).
- `y_2`: Current Y-position of the second bob (pixels).

2.3 `__init__(...)` Constructor Method

Purpose: Initialize a double pendulum with geometry, masses, gravity, and initial state.

Inputs:

- `origin_x`, `origin_y`: Pivot coordinates (pixels). Defaults: 300, 100.
- `length_rod_1`, `length_rod_2`: Rod lengths L_1, L_2 (pixels). Defaults: 120, 120.
- `mass_bob_1`, `mass_bob_2`: Masses m_1, m_2 (a.u.). Defaults: 10, 10.
- `g`: Gravity (pixels/s^2). Default: 9.81.
- `theta_1`, `theta_2`: Angles from vertical (rad). Default: $\frac{\pi}{2}$, $\frac{\pi}{2}$.
- `omega_1`, `omega_2`: Angular velocities (rad/s). Default: 0.0, 0.0.

Output: None.

2.4 `step`(dt: float) Numerical Integration Method

Purpose: Advance the system by one timestep of length `dt` by updating $\omega_1, \omega_2, \theta_1, \theta_2$ and recomputing $(x_1, y_1), (x_2, y_2)$.

Input:

- `dt`: Timestep (seconds). Default: 0.06.

Output: None.

Represents: Numerical integration of angular accelerations α_1, α_2 derived from the Lagrangian; updates state forward in time.

2.5 `get_coords`(None) Accessor Method

Purpose: Provide current bob positions for rendering or APIs.

Input: None.

Output: Two-element list of dictionaries with keys `'x'` and `'y'` which can be used for JSON serialization.

$$[\{x\colon x_1,\ y\colon y_1\},\ \{x\colon x_2,\ y\colon y_2\}]$$

Represents: Snapshot of instantaneous Cartesian positions under this models convention of y being downward.

3 Rendering Animation (`index.html`)

The animation renders on an HTML5 canvas using a decoupled update–render loop:

1. **Initialization:** The canvas reads the pivot (x_0, y_0) from data attributes injected by Flask.
2. **State polling (20 Hz):** An async task periodically fetches JSON coordinates from `/coords` and stores them as `latestCoords`.
3. **Render loop (display rate):** `requestAnimationFrame` drives a loop that:
   (a) Updates two trail buffers with the newest bob positions (capped length).
   (b) Clears the canvas and draws the pivot, rods (pivot→bob1→bob2), and the two bobs.
   (c) Renders trails for both bobs as semi-transparent polylines.
4. **Separation of concerns:** Polling handles data freshness, while the RAF loop ensures smooth, frame-synced rendering.

References

[1] © Alex Van Doren (2025).