Random Walks with Classes

6.100 LECTURE 22

SPRING 2026

Announcements

- Pset 6 checkoff due Friday 5/1
- All finger exercises due Friday 5/1 at midnight
  - give feedback on AI-generated hints: https://forms.gle/4ZQpMBMPY2ZGxSj8A
- Pset 7 due next Wednesday 5/6

- Exam 3 next Monday 5/4
  - logistics announcement tomorrow
  - last term's exams released after class
  - additional practice released after Wednesday's class
  - more review during Friday recitation

Last time: class inheritance

- **Classes** let us define our own data abstraction
- **Inheritance** allows us to reuse those abstractions in making new ones
- **Rules for looking up names:**
  - if it isn't in an instance object, look within its class/type
  - if it isn't in a type object, look within its super/parent class
- All types in Python inherit from the **`object`** type
- Special **double-underscore (dunder)** names in Python
  - commonly refer to "behind-the-scenes" methods for classes
  - implement them to customize their behavior for your own types
  - e.g., **`__init__()`**, **`__eq__()`**, **`__add__()`**, …

Designing `__eq__()`

- **`Rabbit`** example
  - each **`Rabbit`** has two parents and a unique ID
  - new **`Rabbit`**s are created by **`+`**
  - want siblings from the same parents to compare **`==`**
- Version 1
  - compare **`==`** on parent tuples
  - triggers **`==`** and hence **`__eq__()`** on elements, recursion!
  - if self is a **`Rabbit`** and other is **`None`**, then invalid to access **`other.parents`**
- Version 2
  - directly compare parent IDs
  - avoids recursion, saves computation
  - runs into same problem retrieving **`parent.id`** if **`parent`** is **`None`**
- Version 3
  - so close! need a valid "ID" for a **`None`** parent
  - wrap that concept in a helper function, good opportunity to use **`lambda`**

Random Walks

Modeling “walker” entities

- Wrap location in a **`Walker`** instance
  - initialize with a start location
  - **`step()`** specifies possible deltas
  - **`move()`** updates the location
- Would like to “add” step to location
  - to use **+**, extend built-in list class with custom **`__add__()`**
  - to use **+=**, override incremental add operation **`__iadd__()`**
    - be careful about mutation
- Besides using **`Walker`** and **`Vector`** classes, code from before remains largely identical

Modeling different random walk outcomes

- Subclasses of **`Walker`** can override default **`step()`**
- Before
  - had different names for each step function
    - **`step_uniform(), step_left_bias(), ...`**
  - pass in function to customize random walk behavior
- Now
  - different class names, same **`step()`** method name
  - pass in walker type to create walkers with different step behavior
- Keep **`Walker.step()`** for interface, but raise **`NotImplementedError`**

Modeling gas particles

- Particles suspended in a fluid (e.g., dirt, pollen, viruses) look like they perform random walks – **Brownian motion**
  - https://en.wikipedia.org/wiki/Brownian_motion
  - mechanism: random pushing around by the much smaller particles of the fluid
  - how do the fluid particles move?
- Gas molecules can be modeled as performing random walks, too, just with a different **step function**
  - account for motion in continuous space, not just discrete locations – **`ContinuousWalker`**
  - change direction mainly on collision with other particles or barriers – **`InertialWalker`**
    - save current direction as internal attribute **`self._last_step`**
    - reset as needed upon collision

Simulating a collection of gas particles

- A **`Field`** keeps track of multiple particles
  - hence can store particle locations in a **`Field`**
  - and **`Field`** is also responsible for updating particle locations
    - replace **`Walker.move()`** with **`Field.move_particle()`**
    - Walker particles still specify their own **`step()`**s
    - **ensure that particles stay within the field**
  - somewhat analogous to **`DiseaseSimulation`** class on Pset 7
- Simulation function is straightforward
  - initialize all **`InertialWalker`** particles with random initial locations
  - during each simulation step, move each particle one step
- Modeling collisions
  - in principle, should move all particles simultaneously, so interactions only depend on current positions
  - not handling this can be justified if the step size is much smaller than the "interaction distance"

Verify simulation against Ideal Gas Law

- $pV = nRT$
- n is **number of particles**
- In two dimensions, volume V becomes **area A,** and need to adjust constant factor R
- T is **temperature,** which is a measure of kinetic energy
  - energy scales with **velocity squared v^2**
  - in our simulation, velocity is represented by the **number of steps** in the random walk
- p is **pressure,** which is force per unit area (now perimeter in 2D)
  - force is momentum transfer per unit time
  - since we ***model velocity using the walk length,*** the entire walk takes a fixed "unit of time", so we count the total **number of wall hits**
  - momentum transfer for each collision is proportional to **velocity,** so we include another factor of **walk length**

Next time

- Modeling graphs using classes