Wrap-up functions, Stochastic programs, Simulation

6.100 LECTURE 4

SPRING 2026

Announcements

- Pset 2 released after class
- Pset 1 due Friday 2/13
- Pset 1 checkoff due next Friday 2/20
  - try to do it early next week
- Office hours moved to **36-153 starting today,** same times
- **Next Monday is holiday**
  - no office hours
  - office hours next Tuesday moved to **36-156**
  - Tuesday is a Monday schedule
  - pre-lecture code will still be released Sunday around noon
- **Next Friday 2/20 is last day to switch to 6.100A**
  - look at 6.100A website to see course structure
  - discuss with me at instructor office hours tomorrow or by appointment
- Submit **muddy cards** if you'd like something reviewed at start of next lecture!

More on Functions and Mutation

Funtion call mechanics: review

1. Retrieve function object
2. Evaluate arguments in order
3. **Set up frame** for function call
4. Assign parameter names in frame
5. Run body wrt frame until **`return`**
6. **Remove frame,** and substitute the returned object for the function call

Behavior of `return`

- Recall: **`return`** statement stops function execution
  - return ***`expression`***
- What if leave out expression?
  - returns **None**
- What if no return at all?
  - returns **None** at end of function
- What if return a mutating expression?
  - returns whatever that expression evaluates to
  - could possibly be **None**

Python functions that return None

- Functions with no explicit return actually return **None**
  - a **NoneType** object
  - singleton object: only one instance ever exists in memory
  - comparison with **is** or **is not**
    - examines object identity
    - in contrast, **==** compares object value
- Typically, mutating operations return **None**
  - some_list.**append()**
  - some_list.**extend()**
  - some_list.pop() → *value*
  - some_list.**insert(***index, value***)**
  - some_list.**remove(***value***)**
  - some_list.**clear()**
  - some_list.**reverse()** vs **reversed()** vs **some_list[::-1]**
  - some_list.**sort()** vs **sorted()**
- Be careful about "returning" these calls, often not your intention

When default arguments are mutable

- When a **default argument** is specified for a parameter in a function definition:
  - it is **evaluated when Python creates the function object**
  - the header part of the function object stores a reference to the **default argument object**

- When the function is called without an argument for that parameter:
  - the parameter in the frame gets assigned to that default argument object
  - if that object gets mutated during function execution, it **does not get reset when the function returns**
  - hence, a subsequent call that uses the default argument will start with that mutated object

Stochastic Programs

Why stochastic programs?

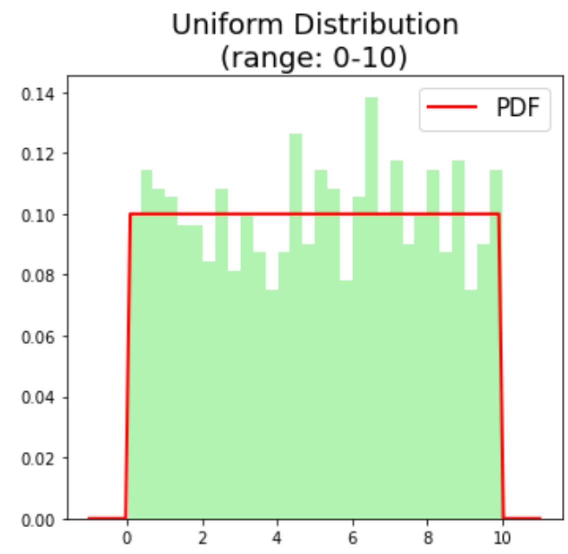
- So far, all the operations we've shown are **deterministic**
  - so given a certain input, a program always produces the same output
- Real life is full of uncertainty!
  - **Predictive nondeterminism:** could perform a deterministic calculation in theory, but lacking input information
    - weather forecasting
    - polling data
  - **Causal nondeterminism:** some events truly random
    - AI text generation
    - outcome of the Super Bowl (before last weekend)
- Value of **simulation**
  - model a process, with uncertainty baked in
  - perform multiple runs/trials of the process to see an ensemble/distribution of possible results

Python's random module

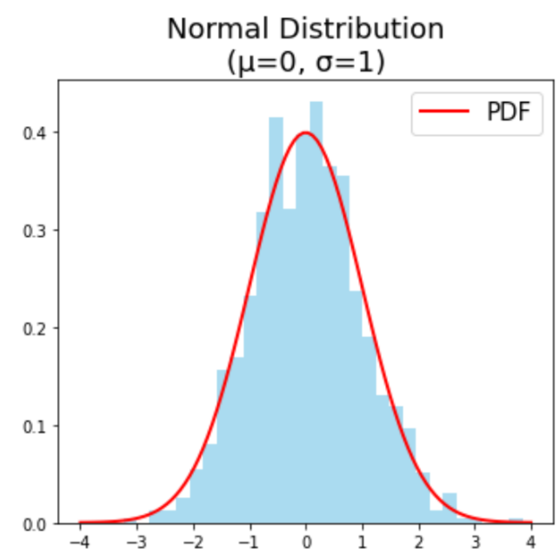
- Library of functions for generating random numbers/data
  - https://docs.python.org/3/library/random.html
    - docs.python.org > Library reference > Numerical and Mathemtical Modules > random
  - use `import random` at top of file
- Basic functionality
  - `random.`**`randint(`***`low, high`***`)`**
  - `random.`**`random()`** → **`float`** between 0 and 1
  - `random.`**`choice(`***`sequence`***`)`**

Sampling from known distributions

- **`random.uniform(`*`low, high`*`)`**
  - https://en.wikipedia.org/wiki/Continuous_uniform_distribution



- **`random.gauss(`*`mu, sigma`*`)`**
  - https://en.wikipedia.org/wiki/Normal_distribution
  - ***mu*** is mean
  - ***sigma*** is standard deviation



Estimating outcome probabilities

- Scenario: **rolling dice**
  - **single die** has $\frac{1}{6}$ chance of landing on side 1
  - **two dice,** if rolled independently, have a $\frac{1}{6} \times \frac{1}{6}$ chance of both landing on side 1
  - **five dice:** $\frac{1}{6^5}$ chance of all landing on side 1
- ~~Lazy~~ *Computational* **way:** roll n dice many times and see how many times they come up all 1's
- **Program decomposition**
  - model rolling a single die
  - model rolling a collection of dice
  - run many trials of rolling a collection

Estimation through random sampling

Estimating pi (π)

- Imagine it's 350 BC in Ancient Greece, and you want to characterize the **area of a circle**
  - https://en.wikipedia.org/wiki/Area_of_a_circle#History
  - ***Indiana Jones*** falls out of the sky and hands you a ThinkPad X1 Carbon Gen 13 Aura Edition (14" Intel) with Python installed
- A circle is fundamentally characterized by its **radius**
  - thus, you reason the area must be proportional to its **radius squared**
  - but what is the **proportionality constant?**
  - ***Pennywise*** falls out of the sky and offers you a pie
  - you politely decline, but it inspires you to name the constant pi (π)

Estimating pi (π)

- Draw a **unit circle** (with radius 1) centered at the origin
- Then circumscribe it with a **bounding square**
  - side length is 2
- Thrown a bunch of darts to land in the square
  - **`random.uniform(-1, 1)`** in each dimension
  - record how many land in the circle
    - distance from origin must not exceed 1
- Multiply **proportion of darts in circle** by **area of square**
  - Congratulations, you've estimated pi (π)!

Estimating integrals

- Example: integrate $\boldsymbol{y = (x-3)^2 + 1}$
  - on the interval $\boldsymbol{x \in [1, 5]}$
- Similar “dart-throwing” process
  - identify the bounding box
  - determine which darts land under the curve
  - multiply proportion by area of box
- Additional considerations
  - handling “negative” area
  - allocating darts more efficiently
  - how “good” is the estimate

Birthday overlap problem

- **What is the probability of at least two people in a group having the same birthday?**
- If there are 30 people in a room, should you be surprised if two share a birthday?
  - **extreme case:** what if there are 367 people?
- Can be solved analytically
  - but not easily amenable to extensions, e.g.:
    - are all birthdays **equally likely?**
    - how likely that there are **three people** who share a birthday?
  - **stochastic simulation** to the rescue!

Next time

- Recitation this Friday: more practice with functions
  - **environment diagrams**
  - **program decomposition**
- Next Tuesday: simulate a type of process known as a **random walk**