Lecture 12:

THE `random` MODULE

CSC111: Introduction to CS through Programming

R. Jordan Crouser
Assistant Professor of Computer Science
Smith College

Announcements

Terry-Ann Craigie

Friday, October 12 • 5:30 p.m.
Seelye Hall, Room 106

Intersections of Race, Data Science and Public Policy

Terry-Ann Craigie, Ph.D., associate professor of economics at Connecticut College, will introduce key public policy issues, describe the limitations of current research and illustrate how big data can help facilitate innovative solutions for improving racial equity.

Her research explores economics of the family, crime and labor. She is currently focusing on equity issues facing the U.S. correctional population, the majority of whom are young racial-ethnic minority males.

Sponsored by the Statistical and Data Sciences Program and the Smith College Lecture Committee.

Free, open to the public and wheelchair accessible. For disability access information or accommodations requests, please call 413-585-2407. To request a sign language interpreter, call 413-585-2071 (voice or TTY) or send email to ods@smith.edu at least 10 days before the event.

SMITH COLLEGE

Overview

- ✓Announcements
- ✓Debrief of “Life Skill #2: Debugging”
- ✓Loops
  - ✓`for…in` (looping through items in a list)
  - ✓the `range()` function (getting a list of numbers)
  - ✓`while` (looping until something happens)
- The `random` module
- Lab: Old MacDonald
- Life Skill #3: Documentation

Recap - Assignment #3: Magic 8 Ball

Discussion

What does it mean

for something to be **random**?

Expectation #1: even distribution

Expectation #1: even distribution

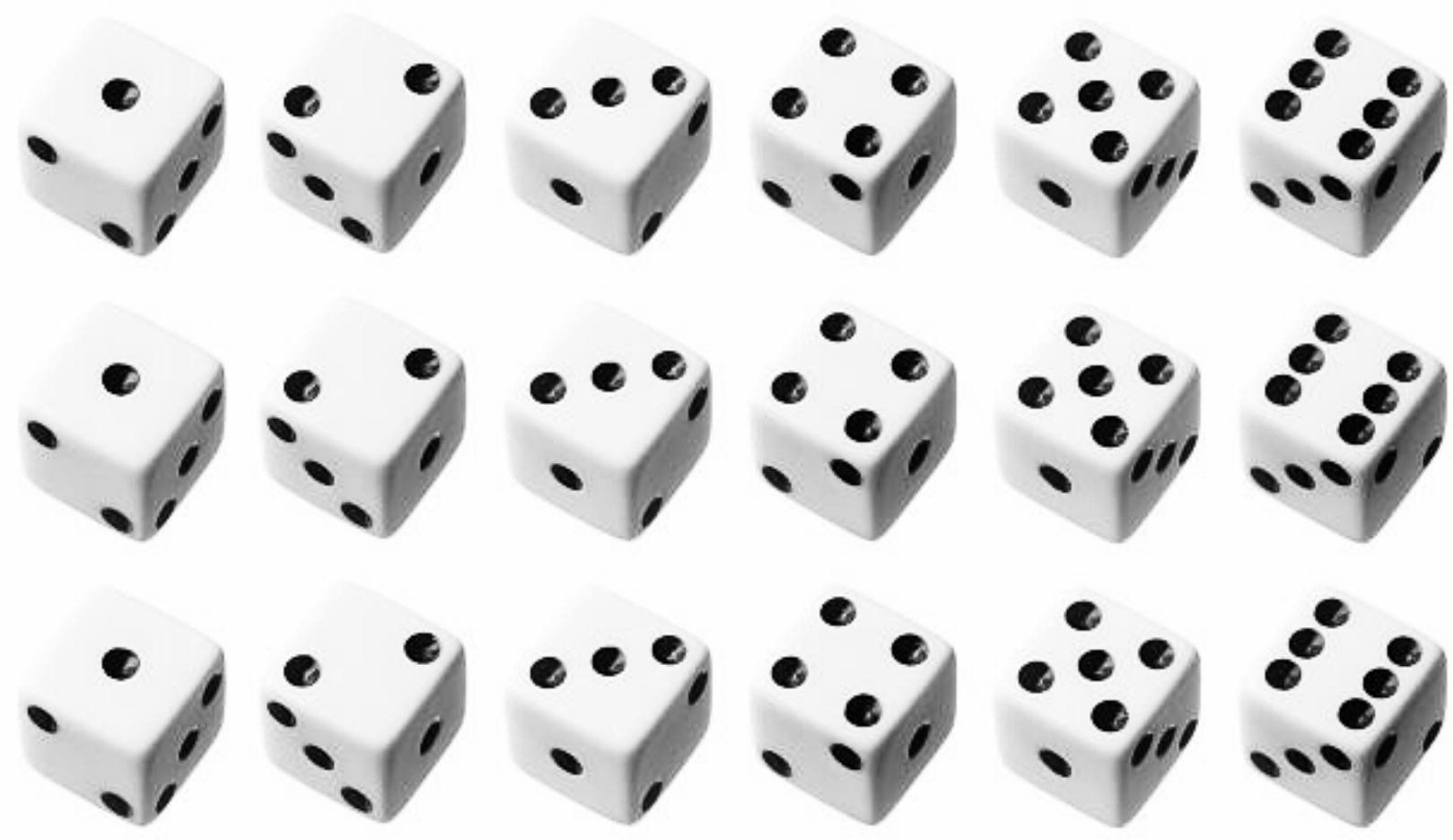
- Every value has an **equal chance** of being chosen
- **Example**: if we roll a die several times, we expect to see:
  - 1 roughly 1/6 of the time
  - 2 roughly 1/6 of the time
  - 3 roughly 1/6 of the time, etc.
- **On average** (over a large number of samples) the distribution is roughly uniform

Discussion

Is an even distribution

enough?

What if the die always rolled like this?



Expectation #2: unpredictable

Expectation #2: unpredictable

- Randomness is more than ensuring that every value has an equal chance of being chosen
- We also want each value to be **hard to predict**
- **Specifically**: seeing several values in the series (“rolls”) shouldn’t help us **guess the next one**

Pseudorandom numbers

pseu·do·ran·dom

/ˌsoōdōˈrandəm/

adjective

(of a number, a sequence of numbers, or any digital data) satisfying one or more statistical tests for randomness but produced by a definite mathematical procedure.

Translations, word origin, and more definitions

"random number generator" (RNG)

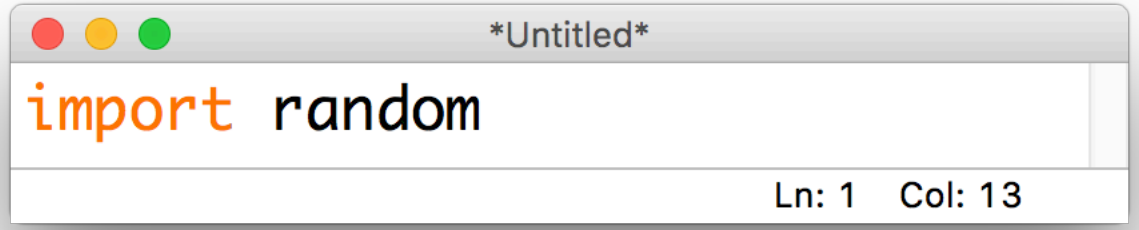
"random enough"

Discussion

How could a **deterministic** machine generate a (seemingly) **random value?**

The `random` module

- Python's built-in RNG can be accessed through the `random` module



- This module contains several useful functions, all of which are documented here:

https://docs.python.org/3/library/random.html

Generating a `random` float

- The simplest way to get a random number is by calling the `.random()` function:

```
*Untitled*
import random
x = random.random()
Ln: 2    Col: 19
```

- This always returns a float with **a value in [0.0, 1)**
- Particularly useful for setting probabilities:

```
if random.random() == 0.7:
    #70% chance of being True
```

10-minute exercise: coin flip

Use the **`.random()`** function from the **`random`** module to write a program that prints **`HEADS`** 50% of the time and **`TAILS`** the remaining 50% of the time

Discussion

What if we wanted a random float

in a different range?

`random` floats in other ranges

- Just use math!
- **Example**: imagine a homework assignment is scored out of 100 points (partial points allowed, and you get 10 points for writing your name)

Untitled

```
import random
# [10.0, 100.0)
score = 10.0 + random.random() * 90.0
```

Ln: 2 Col: 11

Generating a **`random`** integer

- We could multiply, add and call **`int(...)`** to get a random integer using **`.random()`**, but there's no need!
- The **`.randint(...)`** function takes two arguments **`min`** and **`max`**, returns an integer in **`[min, max]`** (inclusive):

Untitled

```
import random
roll = random.randint(1,6)
```

Ln: 2 Col: 26

Discussion

How could we use this to choose a

random item from a list?

Choosing a **`random`** item

- Unsurprisingly, other people have also noticed that this would be a useful feature… so there's a function for that!
- The **`.choice(...)`** function takes in a **`list`**, and returns a randomly selected element:

Untitled

```
import random
courses = ["212", "240", "250"]
spring19 = random.choice(courses)
```

Ln: 3 Col: 32

A common gotcha

- The **`.choice(...)`** function only works when given a **`list-like`** object:
- Don't forget the brackets!

Untitled

```
import random
random.choice("212", "240", "250")
```

Ln: 2 Col: 34

```
TypeError: choice() takes 2 positional
arguments but 4 were given
```

Discussion

What happens if we

call `.choice(...)` on a string?

`.choice(...)` on strings

- Strings are **`list-like!`**
- The “items” in a string are the individual characters, so this is what **`.choice(...)`** chooses between:

Untitled

```
import random
random.choice("ABCDE")
# returns A, B, C, D, or E
```

Ln: 3 Col: 21

A note on testing `random` programs

- It can be really challenging to **test** a program that **behaves differently** every time you run it
- In order to solve this, we can tell python precisely how to generate its (not-so-random-anymore) random numbers using a parameter called a **seed**

*random-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Mat...

```
import random

# Print 10 random numbers
for i in range(10):
    print(random.random())
```

Ln: 3 Col: 25

A note on `import`ing modules

- So far, we've always `import`ed modules like this:

*import-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Mat...

```
import random
import math

random_number = random.random()*100
print(math.sqrt(random_number))
```

Ln: 5 Col: 31

A note on `import`ing modules

- To use a function, we need to specify the **module**:

*import-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Mat...

```
import random
import math

random_number = random.random()*100
print(math.sqrt(random_number))
```

Ln: 5 Col: 31

- This prevents “name clashes” (i.e. if two functions have the same name, the second one overwrites the first)

A note on `import`ing modules

- However, there's also **another way**:

import-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Mater...

```
from random import random
from math import sqrt

random_number = random()*100
print(sqrt(random_number))
```

Ln: 5 Col: 6

- This is useful if we only need specific functions and we want to save ourselves some typing

A note on `import`ing modules

- We can use * to import **everything** from a module :

import-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Mater...

```
from random import *
from math import sqrt

random_number = random()*100
print(sqrt(random_number))
```

Ln: 1 Col: 20

- Again, just be cautious of name clashes…

Overview

- ✓Announcements
- ✓Debrief of “Life Skill #2: Debugging”
- ✓Loops
  - ✓`for…in` (looping through items in a list)
  - ✓the `range()` function (getting a list of numbers)
  - ✓`while` (looping until something happens)
- ✓The `random` module
- Lab: Old MacDonald
- Life Skill #3: Documentation