

Why Does My Computer Do That? Intro to Coding with Python—Random Module

Dr. Ab Mosca (they/them)

Plan for Tuesday

- Next Tuesday, 10/10, is a Monday schedule
- **I will have office hours in this room during class time (9:20 – 11:15)** for anyone who has hw or general questions, or wants to touch base

Plan for Today

- What is random?
- The random module

Discussion

What does it mean
for something to be **random**?

Expectation
#1: even
distribution



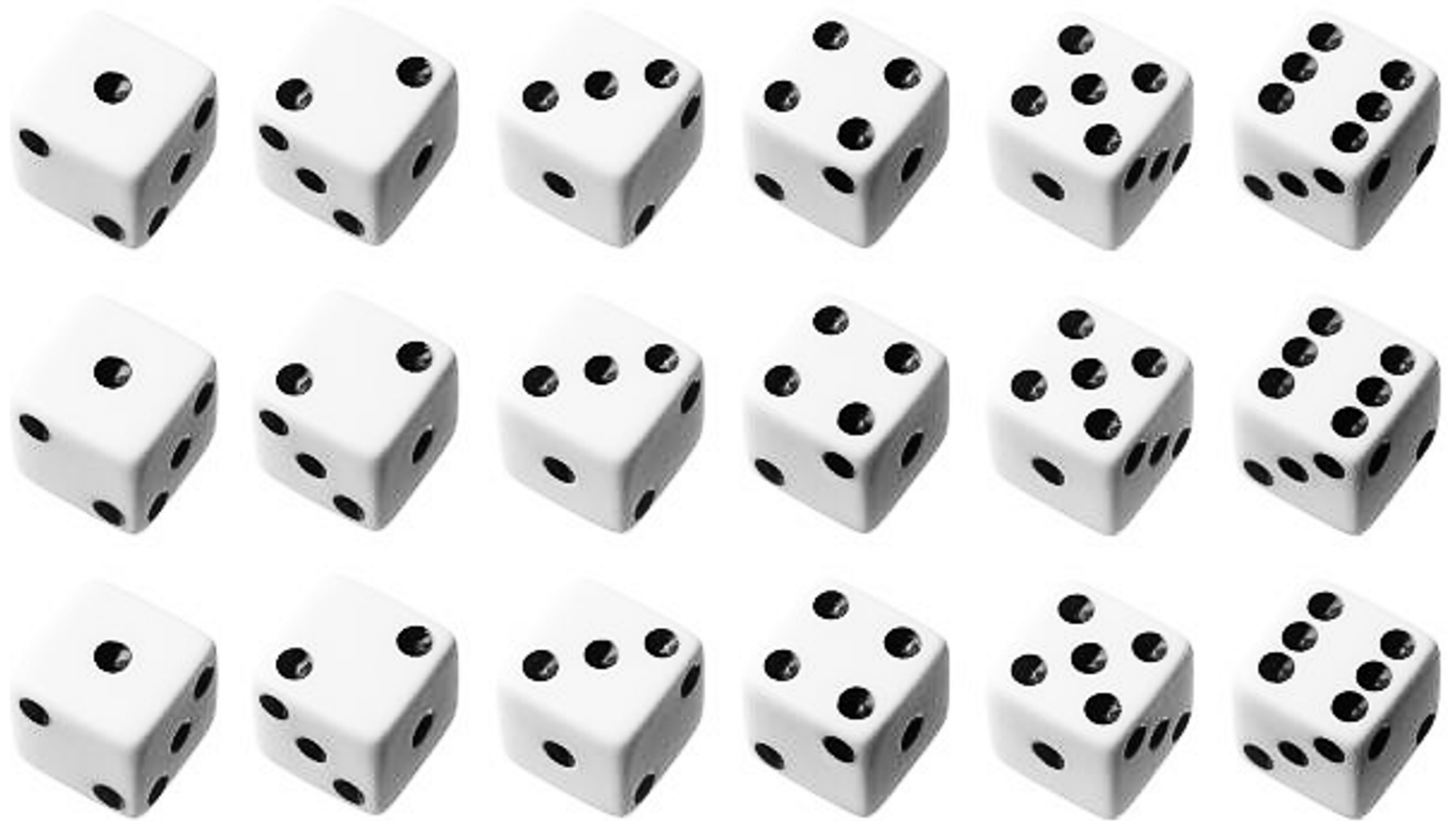
Expectation #1: even distribution

- 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

/ˌsōdōˈrændəm/ 

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

Pseudorandom numbers

pseu·do·ran·dom

/ˌsoʊdōˈrændəm/ 

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.


“random number
generator” (RNG)



Translations, word origin, and more definitions

Pseudorandom numbers

pseu·do·ran·dom

/ˌsoʊdōˈrændəm/ 

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.

“random number
generator” (RNG)



Translations, word origin, and more definitions

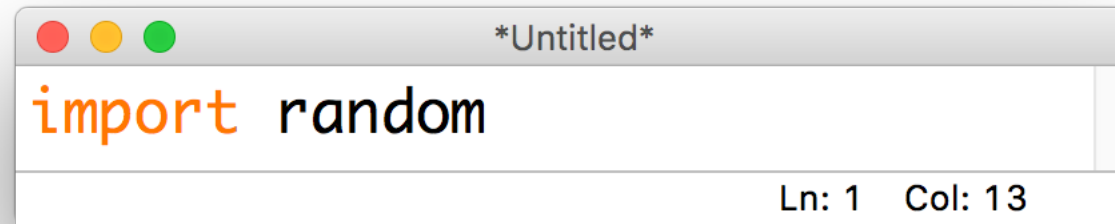
“random enough”

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



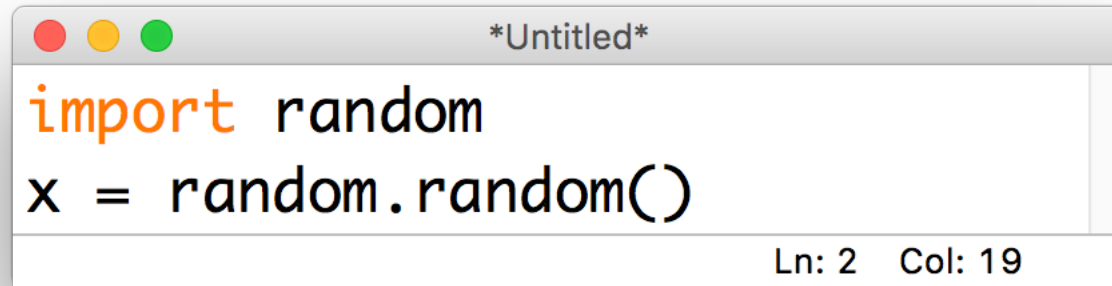
```
*Untitled*  
import random  
Ln: 1 Col: 13
```

- 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

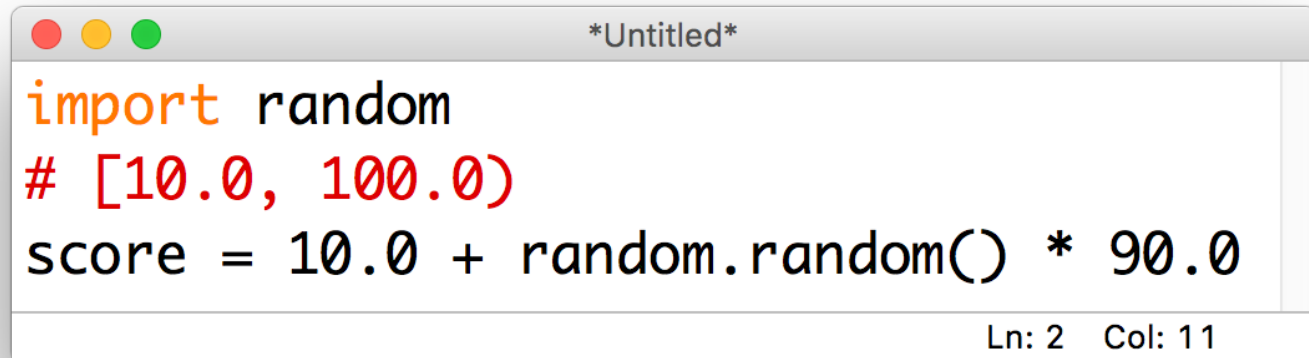


Discussion

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)




```
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):



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

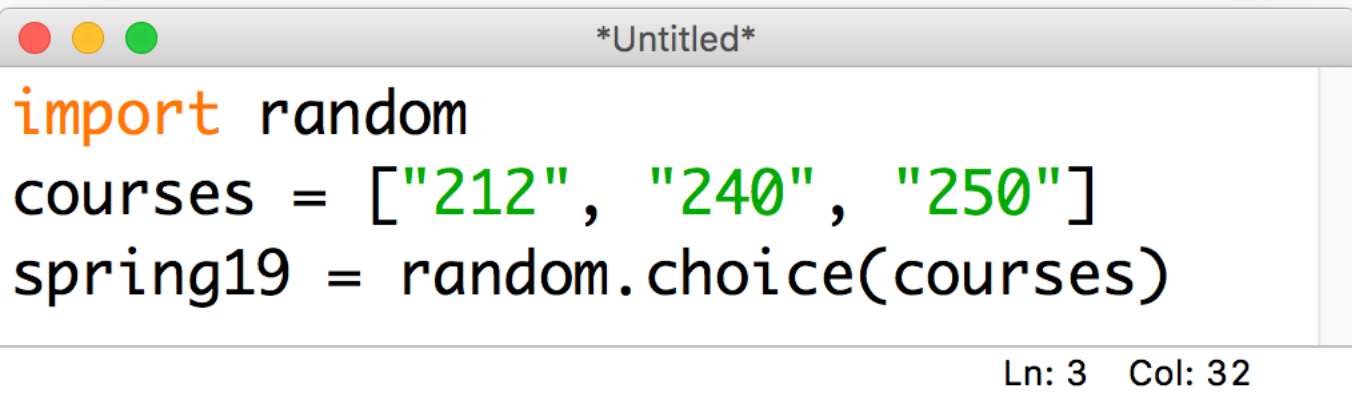
Ln: 2 Col: 26

Discussion

How could we use this to choose a
random item from a list?

Choosing a **random** item

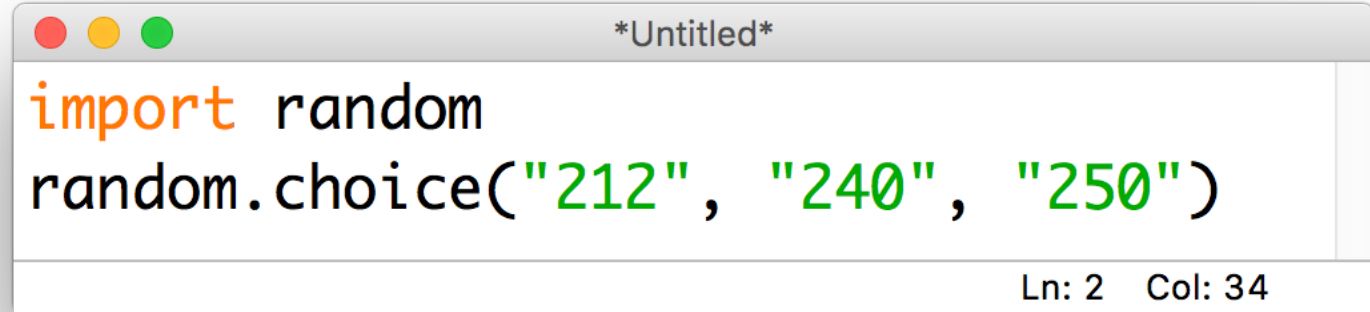
- Unsurprisingly, other people have also noticed that this would be a useful feature... so there's a function for that!
- The `.choice(...)` function takes in a **list**, and returns a randomly selected element:



```
*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!



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

Ln: 2 Col: 34

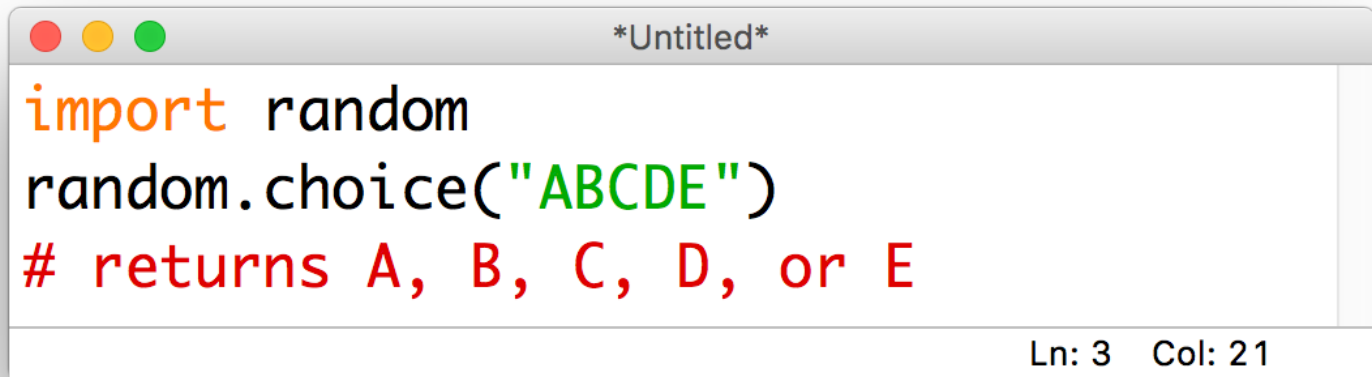
TypeError: choice() takes 2 positional arguments but 4 were given

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:

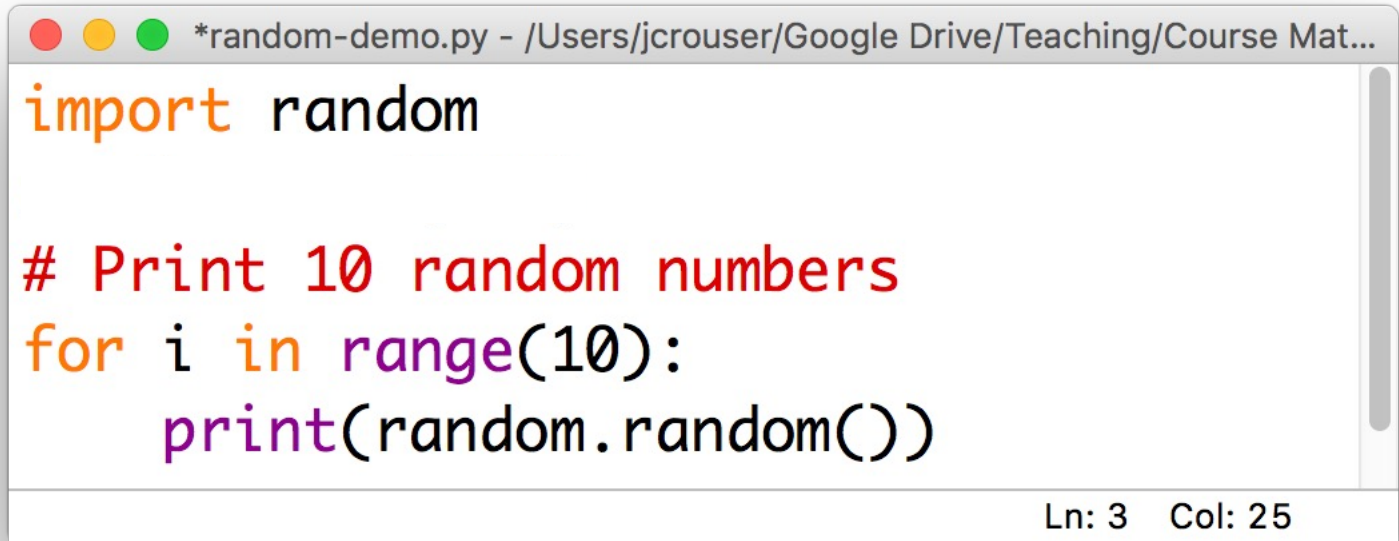


```
import random
random.choice("ABCDE")
# returns A, B, C, D, or E
```

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**

A screenshot of a text editor window titled '*random-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Mat...'. The window contains Python code for generating random numbers. The code is as follows:

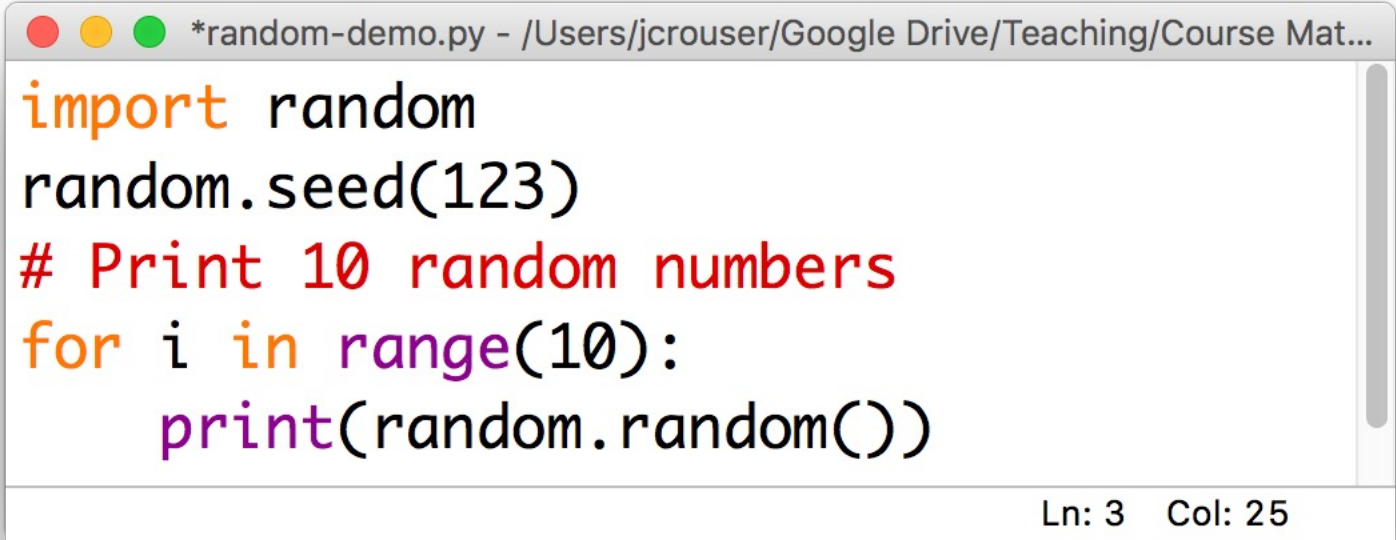
```
import random

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

The code is color-coded: 'import' is orange, 'random' is black, '#' is red, 'Print' is red, '10' is red, 'random' is black, 'numbers' is black, 'for' is orange, 'i' is black, 'in' is orange, 'range' is purple, '(10):' is black, 'print' is purple, 'random.random()' is black. The status bar at the bottom right shows 'Ln: 3 Col: 25'.

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**

A screenshot of a text editor window titled '*random-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Mat...'. The window contains Python code for generating random numbers. The code is as follows:

```
import random
random.seed(123)
# Print 10 random numbers
for i in range(10):
    print(random.random())
```

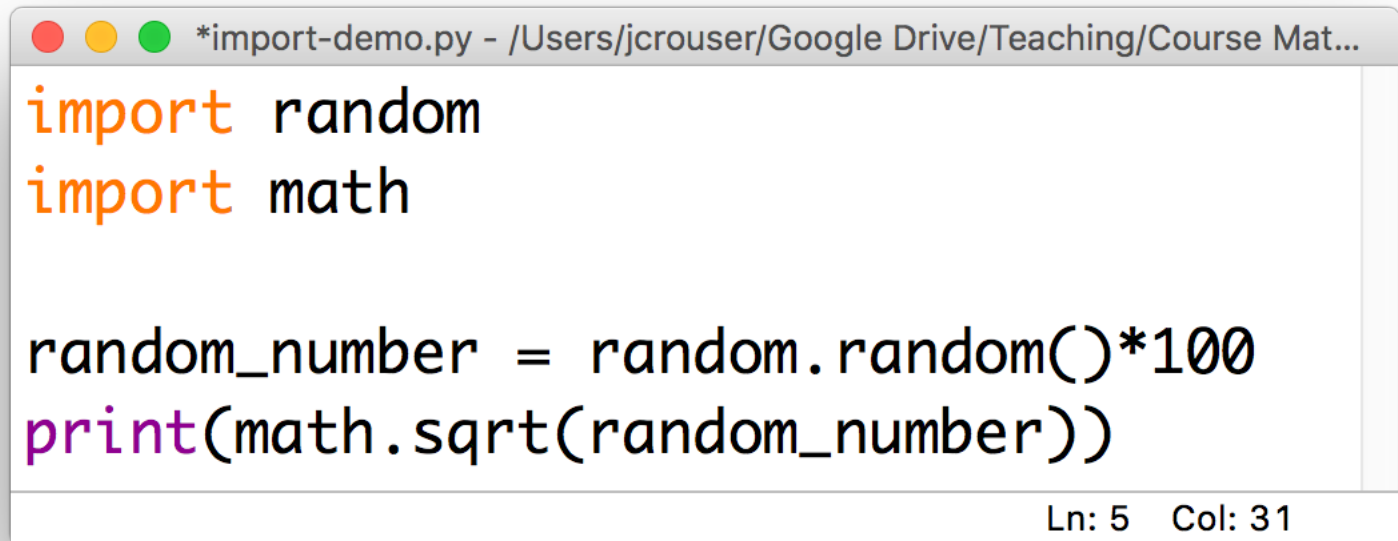
The code is color-coded: 'import' is orange, 'random' is black, 'seed(123)' is black, '#' is red, 'Print 10 random numbers' is red, 'for' is orange, 'i' is black, 'in' is orange, 'range(10):' is purple, and 'print(random.random())' is purple. A scrollbar is visible on the right side of the text area.

```
import random
random.seed(123)
# Print 10 random numbers
for i in range(10):
    print(random.random())
```

Ln: 3 Col: 25

A note on **importing** modules

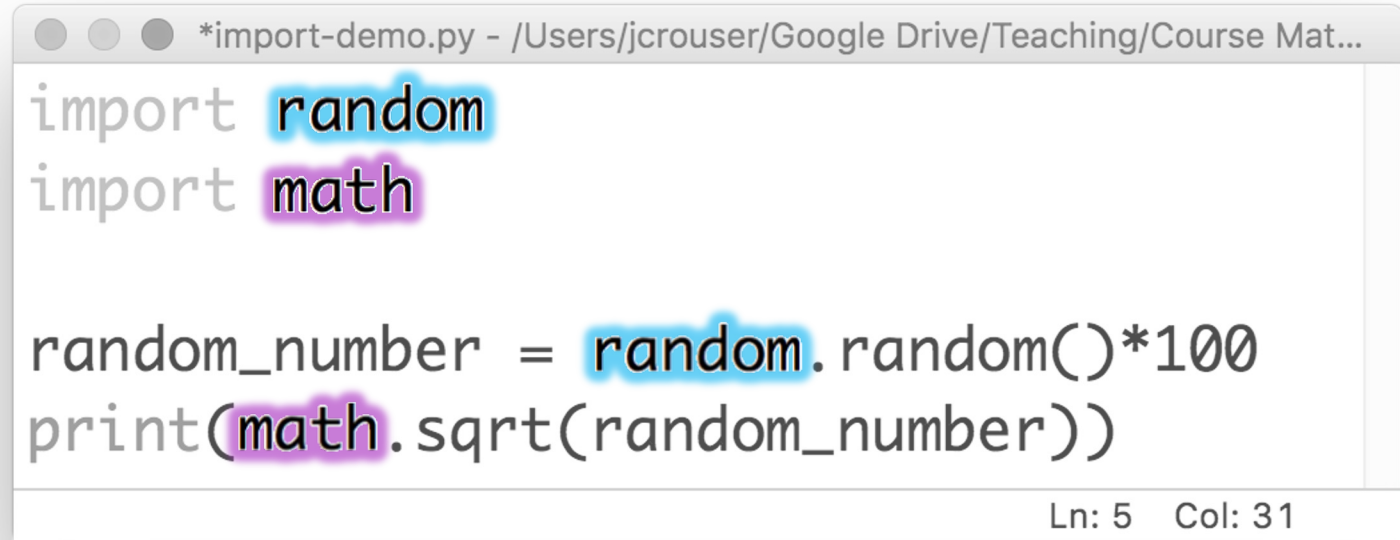
- So far, we've always **imported** modules like this:



```
*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 **importing** modules

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

A screenshot of a code editor window titled '*import-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Mat...'. The code inside the editor is as follows:

```
import random
import math

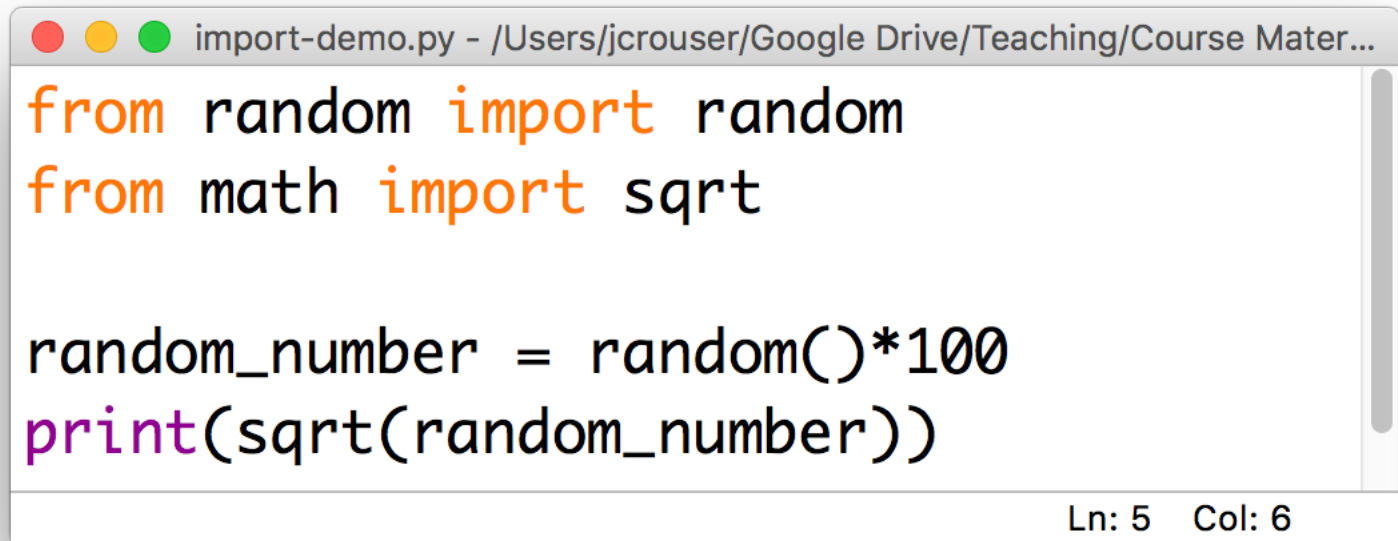
random_number = random.random()*100
print(math.sqrt(random_number))
```

The word 'random' in the first line is highlighted in blue, 'math' in the second line is highlighted in purple, 'random' in the third line is highlighted in blue, and 'math' in the fourth line is highlighted in purple. The status bar at the bottom right of the editor shows '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 **importing** modules

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



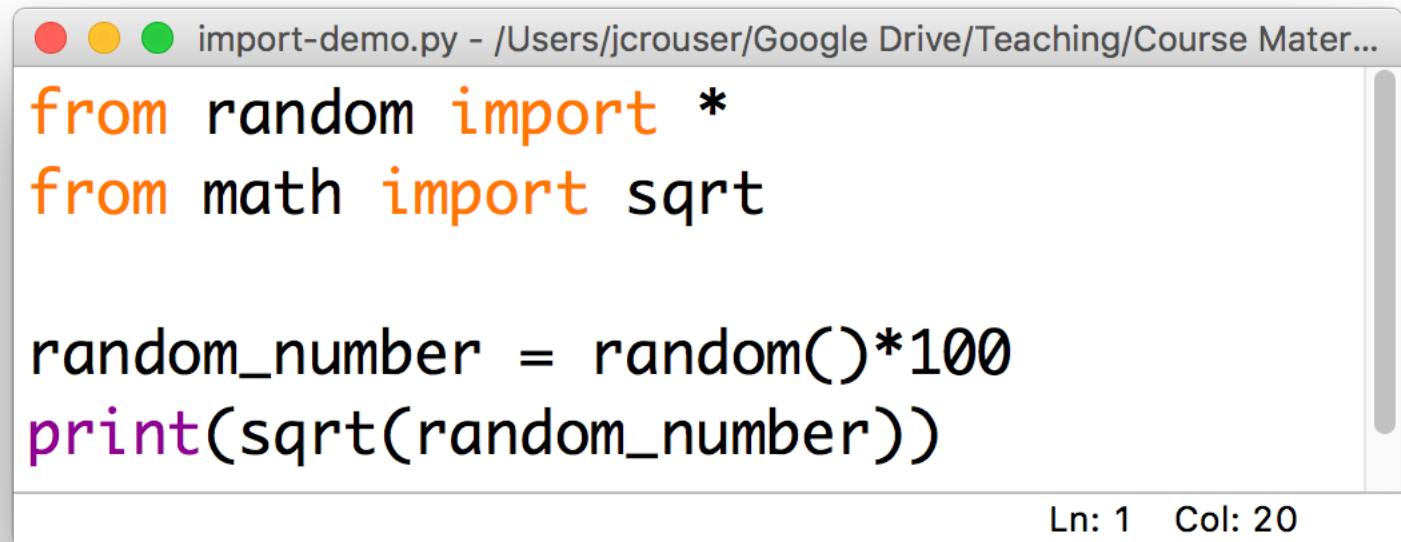
```
import-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Mater...  
from random import random  
from math import sqrt  
  
random_number = random()*100  
print(sqrt(random_number))
```

Ln: 5 Col: 6

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

A note on **importing** modules

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

A screenshot of a code editor window titled 'import-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Mater...'. The window contains Python code with syntax highlighting. The code imports all functions from the 'random' module and the 'sqrt' function from the 'math' module. It then generates a random number between 0 and 100 and prints its square root.

```
from random import *  
from math import sqrt  
  
random_number = random()*100  
print(sqrt(random_number))
```

The status bar at the bottom right of the editor shows 'Ln: 1 Col: 20'.

- Again, just be cautious of name clashes...