# Lec11-modules-RyanSponzilli

October 1, 2024

## 1 ASTR 310 Lecture 11 - Modules

### 1.0.1 Exercise 1: random stuff

The `random` module in the standard library provides random number-related functions, particularly:

- `random.randint(a, b)`: return a random integer between `a` and `b`, inclusive
- `random.random()`: return a random float in the range [0,1)
- `random.shuffle(S)`: randomly shuffle the sequence `S` in place
- `random.choice(S)`: return a random choice from the sequence `S`

Write Python code to do the following. - Create a sorted list of 100 random floats between 0 and $2\pi$. [2 pts]

[1]:
```
import random
import math
numlist = sorted([random.random()*2*math.pi for _ in range(100)])
numlist
```

[1]:
```
[0.05592868409597985,
 0.3506880049319124,
 0.3927058167129476,
 0.4334695592468386,
 0.5467897643136785,
 0.549937923781162,
 0.5998485749174658,
 0.6057983908722628,
 0.788877254470303,
 0.8408963246178985,
 0.885220460257444,
 0.9194558562713665,
 1.0312996175986326,
 1.0861954555137703,
 1.2163902548386978,
 1.274236683731975,
 1.3522395501247464,
 1.357784431001434,
 1.4427490319942011,
 1.4539056404685218,
```

1.5215536209049692,
1.6044951070773172,
1.6065050070047515,
1.6877095350347804,
1.7260497004283841,
1.7516451204681642,
1.75441149724653,
1.8114438326577509,
1.8334740021680602,
1.8367300354951932,
1.8880079474661862,
1.9052925002820005,
1.9662535489284834,
2.0186957945123503,
2.112556861033467,
2.1559844446187486,
2.2452896412195686,
2.2474218976296987,
2.2564049689114354,
2.306970614566529,
2.333941403727375,
2.4759314605876583,
2.4957736780874162,
2.500876090241881,
2.5028170416692133,
2.546817507470503,
2.615124796565787,
2.7672717731958634,
2.7719114380967134,
2.789899221874532,
2.906499478048377,
3.0082792478340705,
3.052440565765505,
3.092092204501085,
3.2211868811339226,
3.292430767802015,
3.3272102502062455,
3.450617373873754,
3.4834251125073257,
3.500157252940817,
3.5453482065657007,
3.690571632780904,
3.871823643983532,
3.955279289320344,
3.979069283340344,
4.125799577756838,
4.16644317227338,

```
4.181167205486836,
4.189732505069648,
4.234113184549845,
4.25620063274743,
4.3642535595505185,
4.4075180684854125,
4.540841670185725,
4.603662792034453,
4.627818661232441,
4.654854251429078,
4.707929966633662,
4.8417339288065,
4.856060778456914,
4.871586966860793,
4.913245810046082,
4.951075826513523,
4.960957215650332,
5.047536511130226,
5.065478098802577,
5.287403076464738,
5.38545071874937,
5.523202257896342,
5.65751520695659,
5.678667546987713,
5.734189693025882,
5.818031395965091,
5.836244368206549,
5.905775826525414,
6.052499056520294,
6.211900746114478,
6.214677222558817,
6.2458208348152064,
6.270497657122542]
```

- Generate a six-digit random integer code (convert to string and pad with zeros if it is less than 100000). [2 pts]

[2]:
```
num = str(int(random.random()*1000000)).zfill(6)
num
```

[2]:
```
'009098'
```

- Generate a ten-character random string containing characters from a set including upper- and lowercase letters, digits, and `#!$@%^&*`. You may like to check helpful information in the documentation for the `string` library. [2 pts]

[73]:
```
codes = list(range(65, 91)) + list(range(97, 123)) + list(range(48,58)) +␣
↪list([35,33,64,37,94,38,42])
```

```
random_chars = [chr(codes[int(random.random()*len(codes))]) for _ in range(10)]
random_string = "".join(random_chars)

random_string
```

[73]:
```
'mGquV7^!zT'
```

### 1.0.2 Exercise 2: time

The `time` module in the standard library provides, unsurprisingly, time-related functions, in particular:

- `time.time()`: return the number of seconds since January 1, 1970 00:00:00 UTC (known as the epoch)
- `time.localtime()`: return a `time_struct` object containing the local time
- `time.asctime(ts)`: given a `time_struct` object `ts`, return a string-formatted date and time
- `time.perf_counter()`: return seconds of a highly accurate counter
- `time.sleep(n)`: pause for `n` seconds

Download the Sieve of Eratosthenes module (`sieve.py`) from the course Canvas page. This contains a single function, `getprimes(n)`, which returns a list of all primes up to `n`. You can import this function into your notebook with `import`.

Write a Python function to accept a value of `n`, then time a call to `getprimes(n)` and return a tuple containing the output of the function and the elapsed time.

If you get bored: make your timing function work with generic functions with arbitrary arguments. [5 pts]

[70]:
```
import sieve
import time

def timeprimes(n):
    start = time.perf_counter()
    p = sieve.getprimes(n)
    stop = time.perf_counter()
    return (p, (stop-start))

timeprimes(1000)
```

[70]:
```
([1,
  2,
  3,
  5,
  7,
  11,
  13,
  17,
  19,
  23,
```

```
29,
31,
37,
41,
43,
47,
53,
59,
61,
67,
71,
73,
79,
83,
89,
97,
101,
103,
107,
109,
113,
127,
131,
137,
139,
149,
151,
157,
163,
167,
173,
179,
181,
191,
193,
197,
199,
211,
223,
227,
229,
233,
239,
241,
251,
257,
263,
```

269,
271,
277,
281,
283,
293,
307,
311,
313,
317,
331,
337,
347,
349,
353,
359,
367,
373,
379,
383,
389,
397,
401,
409,
419,
421,
431,
433,
439,
443,
449,
457,
461,
463,
467,
479,
487,
491,
499,
503,
509,
521,
523,
541,
547,
557,
563,

569,
571,
577,
587,
593,
599,
601,
607,
613,
617,
619,
631,
641,
643,
647,
653,
659,
661,
673,
677,
683,
691,
701,
709,
719,
727,
733,
739,
743,
751,
757,
761,
769,
773,
787,
797,
809,
811,
821,
823,
827,
829,
839,
853,
857,
859,
863,

```
 877,
 881,
 883,
 887,
 907,
 911,
 919,
 929,
 937,
 941,
 947,
 953,
 967,
 971,
 977,
 983,
 991,
 997],
0.0001067080011125654)
```

### 1.0.3 Exercise 3: directory listing

Write a function that uses the `sys` and `os` modules to generate a directory tree listing. The program should take one argument, the name of a directory, and recursively print the names of all files and directories below it. Try to produce output that looks like the following, except, of course, use one of your own directories rather than "foo".

```
foo
|-- a.txt
|-- b.txt
|-- code
|-- |-- a.py
|-- |-- b.py
|-- |-- docs
|-- |-- |-- a.txt
|-- |-- |-- b.txt
|-- |-- x.py
|-- z.txt
```

Hints: `sys.argv`, `os.listdir`, and `os.path.isdir` might be helpful. Also, you may find it easiest to write a recursive function. Try just traversing the directory structure before you print the fancy stuff (indents with "|–" ).

[9 pts]

[68]:
```
import os

def trace_dir(path: str, lvl = 0):
    items = os.listdir(path)
```

```
    for item in items:
        new_path = os.path.join(path, item)
        if os.path.isdir(new_path):
            print("|-- "*lvl + item)
            trace_dir(new_path, lvl=lvl+1)
        else:
            print("|-- "*lvl + item)

trace_dir('/Users/ryansponzilli/Developer/UIUC/astro310/')
```

```
Labs
|-- Lec09-functions-template.ipynb
|-- Lec10-exceptions-RyanSponzilli.pdf
|-- .DS_Store
|-- Lec04-Astro310-09-05-24.ipynb
|-- Lec09-functions-RyanSponzilli.pdf
|-- Lec07-lists-RyanSponzilli.pdf
|-- Lec10-exceptions-template.ipynb
|-- Lec05-varstypes-Ryan-Sponzilli.ipynb
|-- Lec06-control-Ryan-Sponzilli.ipynb
|-- Lec11
|-- |-- sieve.py
|-- |-- Lec11-modules-template.ipynb
|-- |-- __pycache__
|-- |-- |-- sieve.cpython-312.pyc
|-- exercises
|-- |-- .DS_Store
|-- |-- 20240827
|-- |-- |-- .DS_Store
|-- |-- |-- blargh2
|-- |-- |-- |-- foo
|-- |-- |-- ex3.txt
|-- |-- |-- ex2.txt
|-- |-- |-- ex1.txt
|-- |-- |-- blargh1
|-- |-- 20240903
|-- |-- |-- .DS_Store
|-- |-- |-- ex3.in
|-- |-- |-- ex3.py
|-- |-- |-- ex3.txt
|-- |-- |-- ex2.txt
|-- |-- |-- ex1.txt
|-- Lec06-control-Ryan-Sponzilli.pdf
|-- Lec07-lists-template.ipynb
|-- Lec8
|-- |-- Lec08-strings-template.ipynb
|-- |-- table.out
|-- |-- Lec08-strings-RyanSponzilli.pdf
```

```
.DS_Store
Homework
|-- HW5
|-- |-- HW5-template.ipynb
|-- |-- vardict.txt
|-- HW4
|-- |-- Astro 310 HW4.pdf
|-- |-- Astro 310 HW4.ipynb
|-- |-- stars.txt
|-- HW3
|-- |-- Astro 310 HW3.ipynb
|-- |-- primes.txt
|-- |-- Astro 310 HW3.pdf
|-- .DS_Store
|-- Astro 310 HW2.ipynb
|-- Astro 310 HW2.pdf
|-- Astro 310 HW1.docx
```