In_Class_Exercises_11

February 21, 2024

Name: Eklavya Tyagi Net ID: etyagi2

0.0.1 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: - create a sorted random list of 100 floats between 0 and 2 - generate a six-digit random integer code (convert to string and pad with zeros if it is less than 100000) - 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. [6 pts]

[ ]:
```
#Code Here

import math
import random
import string
random_val = [random.uniform(0, 2 * math.pi) for _ in range(100)]
random_val.sort()
print(random_val)

random_int_code = str(random.randint(100000, 999999))
char = string.ascii_letters + string.digits + "#!$@%^&*"
rand_string = ''.join(random.choice(char) for _ in range(10))
print(rand_string)
```

```
[0.09368264424721753, 0.237882402621867, 0.3665614952092273, 0.4476305748510021,
0.6047576208570021, 0.6659337090488311, 0.6780144855990825, 0.6837140821787633,
0.690338946977371, 0.6998370887589301, 0.7473908766668496, 0.851902714534055,
0.8814071346656348, 0.8979304683865683, 0.8985691368133095, 0.9523844598558999,
1.029809359062137, 1.063296228781826, 1.142068854820129, 1.2679897888253167,
1.2791428180547932, 1.2929587405372849, 1.3573254293187877, 1.3882156900830396,
1.448978368798809, 1.4827603709122195, 1.5212677101372438, 1.5792284441020816,
1.6132368114059534, 1.6132958190282838, 1.6386063538408933, 1.6816222079899048,
1.7116597596694647, 1.7336520420581538, 1.8124722501514643, 1.8317153665290027,
```

```
1.8576306417924902, 1.9465448302637287, 1.9859849372675222, 1.9990637107546194,
2.107386530055865, 2.1954877529338193, 2.215333820104513, 2.2615956386529037,
2.4293182625016305, 2.4788493768107918, 2.508027524367379, 2.590950174581247,
2.6312006629971902, 2.7808325013723927, 2.798879786869712, 2.8571835664326173,
2.9206460103641416, 3.0189319214012467, 3.1280786595151575, 3.192564406376732,
3.2099505313029306, 3.2318409967843804, 3.2655718876688464, 3.2876472693034637,
3.4277864977331083, 3.4364282679469724, 3.46952835644826, 3.48781573235493,
3.528943248224344, 3.584884655489212, 3.612101727537313, 3.6176627505290684,
3.791437439921529, 3.9062118667808283, 4.016806193824358, 4.355465747465414,
4.460729660678214, 4.463991326258174, 4.614329583740998, 4.840046380384603,
4.840180313127031, 4.914770191765806, 4.957162909738967, 4.990160032345206,
5.182518346109825, 5.416480379631825, 5.445382009453445, 5.460869659198714,
5.474403431169275, 5.561618884626423, 5.564399872136646, 5.585283815010471,
5.719815025008267, 5.772421518892816, 5.786597473165143, 5.808565376458642,
5.830448415300425, 5.858452105090994, 5.865205482507519, 5.983521253992274,
6.056478171272298, 6.062608873032016, 6.092735932380055, 6.115594982978744]
$5OHu2*HD7
```

0.0.2 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*]

[ ]:
```
#Code Here
import time
from sieve import getprimes

def time_fn(func, *args, **kwargs):
    start = time.perf_counter()
    result = func(*args, **kwargs)
    end_time = time.perf_counter()
    elapsed = end_time - start
    return result, elapsed
```

```
n = 100
primes, elapsed = time_fn(getprimes,n)

print(f"Found {len(primes)} primes up to {n} in {elapsed} seconds.")
```

```
Found 26 primes up to 100 in 1.7099999240599573e-05 seconds.
```

0.0.3 Exercise 3: directory listing

Write a standalone Python program (not a Jupyter notebook) 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 like the following:

foo
|– a.txt
|– b.txt
|– code
| |– a.py
| |– b.py
| |– docs
| | |– a.txt
| | |– b.txt
| |– x.py
|– z.txt

[9*pts*]

[ ]:
```
#Code Here

import os
import sys

def print_dir_tree(startpath):
    for root, dirs, files in os.walk(startpath):
        level = root.replace(startpath, '').count(os.sep)
        indent = ' ' * 4 * (level)
        print(f"{indent}|-- {os.path.basename(root)}/" if root != startpath␣
 ↪else f"{os.path.basename(root)}/")
        subindent = ' ' * 4 * (level + 1)
        for f in files:
            print(f"{subindent}|-- {f}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python dir_tree.py [directory]")
        sys.exit(1)
    directory = sys.argv[1]
```

```
    if not os.path.isdir(directory):
        print(f"Error: {directory} is not a valid directory")
        sys.exit(1)
    print_dir_tree(directory)
```

```
Error: --f=c:\Users\eklav\AppData\Roaming\jupyter\runtime\kernel-v2-
46024ZQUek7MdpGIo.json is not a valid directory
```

```
An exception has occurred, use %tb to see the full traceback.

SystemExit: 1
```

```
The Kernel crashed while executing code in the current cell or a previous cell.


Please review the code in the cell(s) to identify a possible cause of the␣
 ↪failure.

Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for more info.


View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further details.
```

HELLO astro_310/ |– HWs/ |– ASTRO_310_4.ipynb |– HW4_ASTR_310.ipynb |– .git/ |– COMMIT_EDITMSG |– config |– description |– HEAD |– index |– hooks/ |– applypatch-msg.sample |– commit-msg.sample |– fsmonitor-watchman.sample |– post-update.sample |– pre-applypatch.sample |– pre-commit.sample |– pre-merge-commit.sample |– pre-push.sample |– pre-rebase.sample |– pre-receive.sample |– prepare-commit-msg.sample |– push-to-checkout.sample |– sendemail-validate.sample |– update.sample |– info/ |– exclude |– logs/ |– HEAD |– refs/ |– heads/ |– main |– remotes/ |– origin/ |– main |– objects/ |– 0a/ |– 24265155e5d41afdf5026c3c4733b8a082cd5f |– 28/ |– 8d44e57e036f1a8cfd8463a71c36efc04ddd24 |– 4a/ |– b3bd51b46b1df065665aaadbfb8e7f3cf16ada |– 52/ |– 2079e0564d02f2f624ff5849553458d613805b |– 8a/ |– df34867e59e12f815cdf7c8818cbaf6d5ac4a0 |– a8/ |– 8cebc66694d167996ec9569721826e2f8bb660 |– b2/ |– 6323c18083a1e8c283e5c6e2ae3899b7f82152 |– info/ |– pack/ |– refs/ |– heads/ |– main |– remotes/ |– origin/ |– main |– tags/ |– labs/ |– lab10/ |– In_Class_Exercises_11.ipynb |– lec11-modules.pdf |– sieve.py |– **pycache**/ |– sieve.cpython-312.pyc |– lab5/ |– lab5.ipynb |– lab5.pdf |– lab6/ |– lab6.ipynb |– lab6.pdf |– lab7/ |– lab7.ipynb |– lab7.pdf |– table.out |– lab8/ |– lec09_ASTR.ipynb |– lec09_ASTR.pdf |– lab9/ |– lec10_ASTR.ipynb |– lec10_ASTR.pdf |– lab_2/ |– lab_3/ |– ASTR310 EX 25.1.24.ipynb |– ASTR310_EX_25.1.24.ipynb |– ASTR310_EX_25.1.24.pdf |– event_confi.pdf |– lab_4/ |– lab4.ipynb |– lab4.pdf

HERE IS THE RESPONSE

[ ]:
```
%%capture
# Here we use a script to generate pdf and save it to google drive.
```

```
# After executing this cell, you will be asked to link to your GoogleDrive␣
 ↪account.
# Then, the pdf will be generated and saved to your GoogleDrive account and you␣
 ↪need to go there to download;

from google.colab import drive
drive.mount('/content/drive')
# install tex; first run may take several minutes
! apt-get install texlive-xetex
# file path and save location below are default; please change if they do not␣
 ↪match yours
! jupyter nbconvert --output-dir='/content/drive/MyDrive/' '/content/drive/
 ↪MyDrive/Colab Notebooks/In_Class_Exercises_11.ipynb' --to pdf
```