Section1

Elliott Ashby

October 13, 2023

1 q1

The while loop multiplies fac by i returning it to itself then prints fac and i then increments i by one until i reaches 10. Essentially printing a pattern where the right number is multiplied by the number on the next row on the left to find the next number on the right side. Hence

1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880

Since i reaches 10 before it is multiplied by fac the while loop ends and a row with i as 10 does not get printed.

2 q2

```
for i in range(1, 11):
    print(i, i**2, i**3)
```

Here we see a some code the loops over a print statement mutating the variable i each time, starting from 1 and going to 10. The print statement print the value, it's square and it's cube.

3 q3

The answers are m and 3 since the 4th letter of the 3rd to last element of the list b is m and the first entry in a is 3.

4 q4

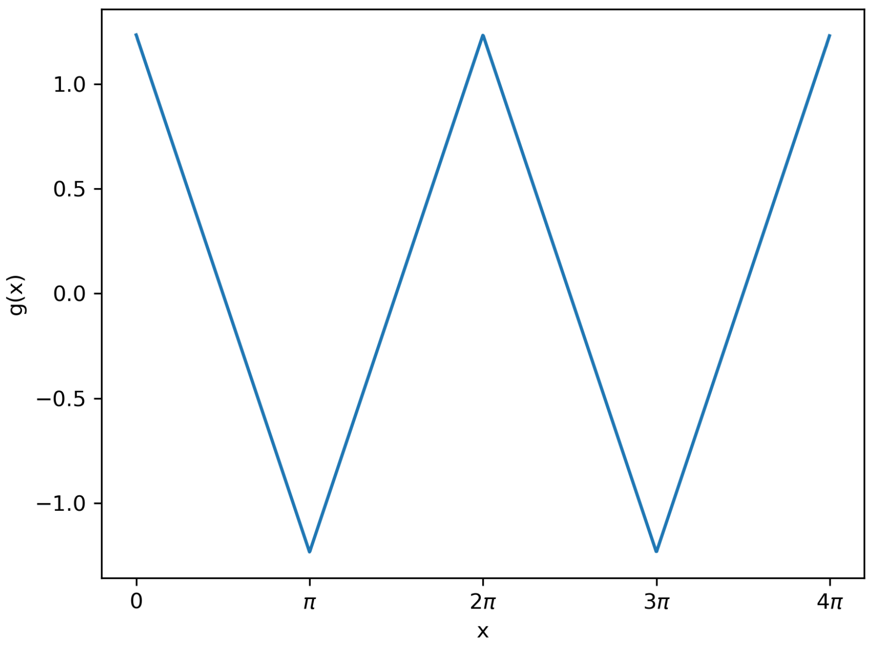
$1e^3$ will take a short time but is less accurate than $1e^{26}$, but $1e^{26}$ takes a long time. $1e^7$ is accurate enought while being still relatively fast.

5 q5

```
import matplotlib.pyplot as plt
import numpy as np
from math import pi
four = __import__('1_4')


if __name__ == "__main__":
    x = np.arange(0, 4*pi, 0.01)
    y = [four.g(i) for i in x]
    plt.plot(x, y)
    plt.xlabel('x')
    plt.ylabel('g(x)')
    plt.xticks(ticks=[0, pi, 2 * pi, 3 * pi, 4 * pi],
        ↪ labels=['0', '$\pi$', '2$\pi$', '3$\pi$', '4$\
        ↪ pi$'])
    plt.savefig('1_5.png', dpi=300)
```

Here we simply create a list of numbers from 0 to 4π in steps of 0.01 and calculate g(x) for each of them. We then plot the list of x values against the list of g(x) values adding axis labels and tick names.



6 q6

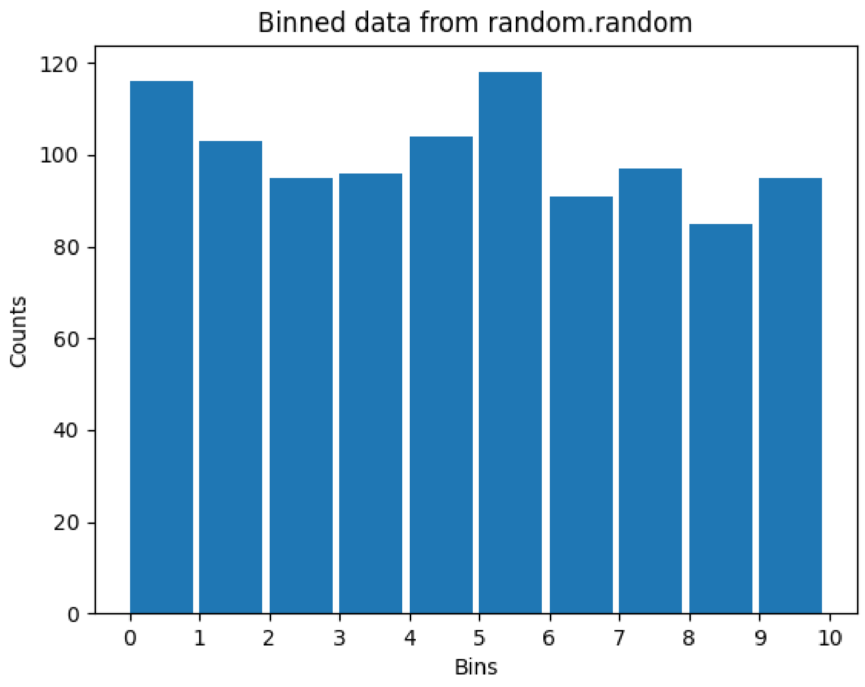
```
from numpy import random
import matplotlib.pyplot as plt


def gen_data(samples: int, _range: int) -> list:
    m: list = []
    for i in range(samples):
        n = random.random() * _range
        m.append(n)
    return m


def sort_to_bins(data, binsNo: int, _range: int) -> list:
    bins = [0 for bins in range(binsNo)]
    for x in data:
        bins[int(x * binsNo)] += 1
    return bins


if __name__ == "__main__":
    samples = 1000
    binsNo = 10
    _range = 1
    binnedData = sort_to_bins(gen_data(samples, _range),
        ↪ binsNo, _range)
    print(binnedData)
    plt.title("Binned data from random.random")
    plt.xticks(range(binsNo + 1), labels = [str(x *
        ↪ _range) for x in range(binsNo + 1)])
    plt.xlabel("Bins")
    plt.ylabel("Counts")
    plt.bar(range(binsNo), binnedData, width=0.9, align='
        ↪ edge')
    plt.savefig("1_6.png")
```

We start by defining our constants, samplesm range and number of bins, we the create the binned data by calling the sort to bins function and the gen data function inside that. We then plot the data as a bar graph with the x axis being the bins and the y axis being the number of samples in each bin.



7 q7

```
from numpy import random


def gen_data(samples: int, _range: int) -> list:
    m: list = []
    for i in range(samples):
        n = random.normal() * _range
        m.append(n)
    return m


def countGreaterThan(data: list, min: float, max: float)
    ↪ -> float:
    count: int = 0
    for num in data:
        if not (min < num < max):
            count += 1
    return count


if __name__ == "__main__":
    samples = 1000000
    _range = 1
    repeats = 5
    count: float = sum([countGreaterThan(gen_data(samples
        ↪ , _range), -2, 2) for _x in range(repeats)]) /
        ↪ repeats
    print(f"{count / samples * 100}% are outside of the
        ↪ range -2 < x < 2")
```

Therefore:

4.53988% are outside of the range $-2 < x < 2$

With obviously some variation in percentage calcuated, however to minimise this the code is ran "repeats" number of times and an average is found.

8 q8

```
import matplotlib.pyplot as plt


def next_node(r: float, s: float, R_i: list) -> float:
    return (r + s * R_i[-1] / (s + R_i[-1]))


def gen_data(r: float, s: float) -> list:
    R_i = [r + s]
    while len(R_i) < 2 or (R_i[-2] - R_i[-1] >= 1e-7):
        R_i.append(next_node(r, s, R_i))
    return R_i


if __name__ == "__main__":
    s = 10000.0
    r = 100.0
    R_i = gen_data(r=r, s=s)
    print(R_i[-1])
    plt.plot(R_i)
    plt.savefig("1_8.png")
```

Here we change the code given to us from a set number of runs "n" to a stopping condition, in this case I chose $1e^{-7}$ since it resulted in a reasonable amount of precision. Here we output the result as:

1051.2492201029809