Lecture01-S25-in-class-edits

January 17, 2025

1 Lecture 1 (pt. 2)

1.1 Basic python operations

1.2 Set operations

1.3 First Random Experiment and Simulations

1.4 Fair experiment and relative frequency

Working with lists:

[1]:
```
a=[2,3]
b=a
a is b
```

[1]:
```
True
```

[2]:
```
b.append(4)
b
```

[2]:
```
[2, 3, 4]
```

[3]:
```
a
```

[3]:
```
[2, 3, 4]
```

You can also append to lists with +:

[4]:
```
b = a.copy()

b is a
```

[4]:
```
False
```

[5]:
```
b.append(5)
b
```

[5]:
```
[2, 3, 4, 5]
```

[6]:
```
a
```

[6]:
```
[2, 3, 4]
```

Lists and tuples may contain any other objects, including other lists and tuples:

[7]:
```
c =(1,2,4)
```

[8]:
```
type(c)
```

[8]:
```
tuple
```

Note that tuples and lists are ordered collections, and we can access their members directly:

[9]:
```
a[0]
```

[9]:
```
2
```

[10]:
```
c[0]
```

[10]:
```
1
```

- Negative indexes start from the end of the list, with -1 denoting the last member in the list:

[11]:
```
c[-1]
```

[11]:
```
4
```

[ ]:
```

```

[ ]:
```

```

1.5 Modules and Libraries

- Many of the tools we will use in the class are not directly part of Python
- Instead, they are libraries or modules that provide particular functionality
- These include:

- **numpy** provides arrays, linear algebra, and math functions (many similar to the core MATLAB functions)
- **matplotlib** provides functions to generate plots similar to those in MATLAB
- **random** contains functions for generating random numbers and choices
- **scipy** provides many tools used in scientific computing including optimization, signal processing, and statistics
- **pandas** provides tools for working with data

To work with these libraries, import them:

[12]:
```
!jt -r  #(reset to default theme in jupyter)
```

```
Reset css and font defaults in:
/Users/Jie/.jupyter/custom &
/Users/Jie/Library/Jupyter/nbextensions
```

```
[13]: # use jupytertheme to change font size and theme (used in lecture only)
      !jt -t grade3 -fs 24  -tfs 24 -ofs 22 -cellw 100%
      #!jt  -fs 12
```

2 Sets

```
[14]: Aset = {1,2,4}
      print(Aset)
```

```
{1, 2, 4}
```

```
[15]: Aset= set([1,1,1,2,3,4])
      print(Aset)
```

```
{1, 2, 3, 4}
```

2.1 Union

```
[16]: Bset= {3,4,5}
      AunionB = set.union(Aset,Bset)
      print(AunionB)
```

```
{1, 2, 3, 4, 5}
```

2.2 set difference

```
[17]: Cset = Aset - Bset
      print(Cset)
```

```
{1, 2}
```

2.3 set intersection

```
[18]: Dset = Aset.intersection(Bset)
      print(Dset)
```

```
{3, 4}
```

see [More about Python Sets]{https://realpython.com/python-sets/}

2.4 import some library for the following experiments

```
[19]: import numpy as np
      import matplotlib.pyplot as plt
      %matplotlib inline
      plt.style.use('bmh')
```

3 First Random Experiment and Simulations

Consider the following questions:

1. **If you flip a coin 20 times, how many times do you think it will come up heads?**
2. **If you flip the coin 20 times and it comes up heads 6 times, do you think it is a *fair* or *unfair* coin? How *confident* can you be in your answer?**

- Can you conduct an experiment to answer these questions?
- What can be a potential problem?

If we take a **fair coin** and flip it 20 times and count the number of heads, and then **repeat the experiment many times**, we can **estimate** how often 6 or fewer heads occurs. If it occurs very rarely (say, less than 5% of the time, then we can say that the coin is unlikely to be fair).

Here we use 6 or fewer heads because if 5 heads occurs, that is an even more extreme outcome than 6 heads occurring, and so we want to count up how often we see an outcome as extreme OR MORE as 6 heads occurring.

- The **problem** is that we may need to repeat the experiment (of flipping the coin 20 times) many times to accurately estimate how often 6 or fewer heads come up. This may require thousands of coin flips!

We can overcome this problem by using a computer to flip the coin in a **simulation**. A **computer simulation** is a computer program that models reality and allows us to conduct experiments that:

- would require a lot of time to carry out in real life
- would require a lot of resources to carry out in real life
- would not be possible to repeat in real life (for instance, simulation of the next day's weather or stock market performance)

Let's build simulations of our coin flip experiment and learn about some Python libraries:

[20]:
```
# Simple library for working with random phenomena
import numpy
import random

# to learn about random.choice, random.choices, and random.randint function
```

[21]:
```
# flip one coin

faces = ['H','T'] # sample space of one coin flip
```

[22]:
```
random.choice(faces)
```

[22]:
```
'H'
```

[23]:
```
?random.choices
random.choices(faces, k=20)
```

```
[23]: ['T',
       'T',
       'T',
       'H',
       'T',
       'H',
       'T',
       'H',
       'H',
       'T',
       'H',
       'T',
       'H',
       'H',
       'T',
       'T',
       'T',
       'T',
       'H',
       'T']
```

```
Signature: random.choices(population, weights=None, *, cum_weights=None, k=1)
Docstring:
Return a k sized list of population elements chosen with replacement.

If the relative weights or cumulative weights are not specified,
the selections are made with equal probability.
File:      ~/opt/anaconda3/lib/python3.9/random.py
Type:      method
```

```
[24]: # Generate a random integer between 1 and 10 (both inclusive)
      random_number = random.randint(1, 10)
      random_number # for die roll experiment
```

```
[24]: 8
```

Suppose we want to see how often 6 or fewer heads occurs. We can reduce the printing by only printing those extreme events:

```
[25]: coins = random.choices(faces, k=20)
      coins.count('H')
```

```
[25]: 9
```

```
[26]: coins.count(0)
```

```
[26]: 0
```

[27]:
```python
# perform repeated experiments
num_sims = 25
flips = 20

for sim in range(num_sims):
    coins = random.choices(faces, k=flips)
    num_heads = coins.count('H')
    print(sim, ': ', num_heads, ' Head')
```

```
0 :  13  Head
1 :  11  Head
2 :  11  Head
3 :  11  Head
4 :  15  Head
5 :  8  Head
6 :  7  Head
7 :  10  Head
8 :  10  Head
9 :  12  Head
10 :  8  Head
11 :  14  Head
12 :  9  Head
13 :  7  Head
14 :  11  Head
15 :  9  Head
16 :  7  Head
17 :  7  Head
18 :  10  Head
19 :  9  Head
20 :  9  Head
21 :  8  Head
22 :  11  Head
23 :  11  Head
24 :  5  Head
```

- We really don't care about the particular experiment on which those events occur. Instead, we are really just looking at the **frequency** of those events.

Relative Frequency

The relative frequency of an event is the number of times that an event occurs divided by the number of times the experiment is conducted.

Let's modify the experiment to calculate the relative frequency of getting 6 or fewer heads on 20 flips of a fair coin:

[28]:
```python
# perform repeated experiments
num_sims = 25
flips = 20
# add a counter: for counting the number of times event occurs:
```

```python
# Event: num of heads <= 6
counts = 0

for sim in range(num_sims):
    coins = random.choices(faces, k=flips)
    num_heads = coins.count('H')
    if num_heads <=6:
        counts += 1 # counts = counts+1

print('In ', num_sims, 'experiments', '<= 6 heads occurred: ', counts)
```

```
In  25 experiments <= 6 heads occurred:  1
```

[29]:
```python
# calculate relative frequency
num_sims = 5000
flips = 20
# add a counter: for counting the number of times event occurs:
# Event: num of heads <= 6
counts = 0

for sim in range(num_sims):
    coins = random.choices(faces, k=flips)
    num_heads = coins.count('H')
    if num_heads <=6:
        counts += 1 # counts = counts+1

print('Relative frequency of <=6 heads is', counts/num_sims)
```

```
Relative frequency of <=6 heads is 0.0598
```

4 Fair Experiment

Example 1:: The probability of getting any number on a fair 6-sided die is 1/6. Let's compare these to the *relative frequencies*.

But first let's see how to count the number of occurrences of each outcome:

[30]:
```python
num_sims=1000
values=[] # output of the die roll.

for sim in range(num_sims):
    die=random.choice(range(1,7)) # range(1,7) =[1,2,3,4,5,6]
    values += [die] # values.append(die)
print(values)
```

```
[2, 1, 3, 6, 2, 1, 3, 5, 1, 6, 5, 3, 6, 6, 4, 4, 5, 4, 5, 6, 1, 2, 2, 2, 4, 1,
5, 6, 5, 2, 4, 3, 5, 6, 6, 3, 4, 3, 4, 5, 2, 6, 3, 6, 1, 6, 2, 4, 4, 3, 1, 3, 1,
5, 6, 4, 5, 2, 5, 4, 3, 2, 1, 2, 1, 4, 5, 2, 1, 3, 1, 2, 1, 5, 3, 4, 6, 5, 6, 6,
6, 3, 2, 1, 4, 3, 5, 4, 5, 6, 5, 5, 4, 1, 5, 6, 4, 6, 5, 4, 2, 2, 3, 3, 2, 1, 2,
4, 4, 5, 1, 1, 1, 2, 5, 4, 1, 2, 5, 2, 1, 5, 5, 6, 3, 2, 6, 6, 5, 5, 3, 5, 2, 4,
5, 1, 3, 5, 4, 5, 6, 3, 3, 5, 3, 6, 3, 3, 2, 4, 4, 2, 6, 6, 5, 4, 3, 1, 3, 2, 6,
2, 3, 6, 2, 2, 1, 5, 3, 6, 6, 4, 6, 4, 2, 1, 4, 3, 2, 4, 6, 3, 3, 4, 3, 4, 2, 6,
4, 6, 1, 6, 6, 5, 3, 2, 3, 2, 2, 3, 5, 5, 2, 5, 3, 3, 2, 1, 1, 1, 5, 1, 6, 4, 3,
6, 5, 5, 2, 5, 2, 6, 4, 5, 3, 2, 2, 2, 3, 1, 6, 1, 6, 1, 4, 2, 2, 4, 2, 4, 5, 1,
1, 1, 6, 3, 5, 6, 6, 4, 3, 6, 6, 4, 2, 2, 5, 3, 4, 1, 4, 4, 5, 2, 4, 5, 1, 6, 6,
1, 1, 5, 3, 2, 1, 4, 2, 6, 6, 2, 3, 2, 2, 1, 1, 4, 3, 5, 2, 1, 6, 3, 6, 4, 2, 5,
5, 2, 5, 4, 6, 4, 4, 3, 1, 2, 2, 4, 2, 2, 6, 1, 4, 1, 4, 4, 3, 6, 1, 5, 5, 3, 2,
5, 4, 2, 2, 6, 3, 1, 2, 1, 3, 6, 1, 5, 1, 6, 4, 6, 5, 3, 6, 3, 6, 4, 1, 4, 2, 2,
1, 2, 3, 2, 5, 6, 1, 4, 2, 3, 1, 4, 3, 2, 4, 4, 4, 5, 6, 5, 2, 1, 3, 1, 1, 2, 2,
1, 3, 6, 1, 6, 5, 3, 6, 6, 4, 3, 4, 4, 4, 1, 5, 3, 2, 4, 1, 6, 1, 3, 6, 3, 4, 3,
3, 6, 6, 1, 4, 3, 6, 4, 1, 2, 4, 2, 5, 4, 1, 4, 3, 2, 5, 4, 1, 3, 1, 6, 1, 5, 2,
1, 1, 1, 5, 5, 4, 6, 2, 1, 3, 3, 4, 3, 1, 6, 3, 4, 4, 2, 3, 5, 6, 4, 2, 4, 1, 1,
4, 6, 1, 2, 5, 3, 3, 6, 6, 2, 4, 6, 6, 5, 1, 6, 4, 2, 6, 3, 1, 4, 4, 5, 6, 5, 5,
4, 4, 4, 6, 1, 6, 6, 6, 5, 1, 5, 4, 6, 6, 6, 3, 1, 5, 4, 1, 5, 6, 1, 1, 6, 6, 5,
5, 2, 2, 4, 3, 1, 6, 6, 2, 6, 2, 1, 1, 4, 3, 5, 2, 5, 1, 5, 5, 5, 4, 3, 3, 1, 3,
1, 3, 6, 5, 4, 1, 2, 3, 4, 2, 6, 3, 2, 6, 6, 1, 5, 1, 1, 3, 2, 3, 2, 1, 1, 3, 5,
5, 4, 5, 2, 5, 1, 5, 5, 3, 5, 1, 6, 3, 5, 4, 1, 6, 2, 4, 4, 2, 5, 2, 5, 1, 6, 1,
6, 3, 6, 6, 3, 2, 3, 4, 1, 1, 5, 5, 3, 5, 1, 2, 5, 2, 3, 3, 3, 3, 6, 3, 4, 3, 4,
6, 4, 4, 3, 6, 6, 5, 5, 6, 2, 6, 5, 2, 2, 4, 6, 5, 6, 5, 5, 1, 4, 2, 2, 5, 5, 4,
4, 1, 3, 1, 5, 6, 4, 4, 5, 5, 4, 6, 2, 6, 1, 3, 2, 6, 5, 2, 5, 3, 6, 6, 2, 2, 2,
1, 1, 2, 6, 6, 3, 3, 5, 5, 5, 2, 4, 5, 5, 6, 5, 4, 3, 3, 5, 1, 3, 4, 5, 3, 5, 5,
6, 6, 5, 3, 1, 6, 6, 2, 1, 1, 2, 6, 5, 3, 3, 5, 4, 4, 5, 4, 2, 3, 5, 3, 1, 4, 3,
4, 4, 3, 4, 6, 2, 1, 2, 3, 5, 6, 2, 1, 1, 4, 5, 4, 6, 6, 5, 3, 6, 4, 6, 5, 1, 2,
6, 2, 2, 5, 3, 6, 4, 3, 2, 5, 6, 6, 2, 5, 4, 5, 3, 3, 4, 4, 5, 6, 2, 6, 2, 2, 4,
5, 2, 4, 2, 3, 2, 4, 6, 6, 5, 4, 4, 2, 3, 5, 3, 4, 3, 4, 4, 3, 3, 6, 3, 5, 1, 5,
6, 5, 5, 2, 4, 5, 2, 1, 1, 5, 2, 4, 4, 5, 5, 3, 3, 4, 4, 1, 4, 4, 3, 4, 3, 1, 5,
1, 1, 4, 4, 6, 5, 5, 4, 2, 2, 1, 5, 1, 3, 4, 4, 4, 5, 1, 3, 5, 5, 5, 4, 5, 1, 4,
1, 2, 5, 5, 4, 3, 2, 5, 3, 5, 1, 6, 4, 3, 4, 6, 1, 6, 6, 4, 5, 3, 6, 5, 6, 4, 5,
6, 6, 3, 4, 2, 3, 6, 1, 6, 1, 3, 5, 4, 5, 1, 6, 4, 4, 2, 1, 3, 2, 3, 5, 5, 1, 2,
1, 4, 6, 3, 5, 3, 6, 3, 4, 4, 1, 1, 5, 1, 6, 5, 1, 4, 1, 2, 4, 5, 6, 1, 5, 3, 1,
1, 6, 1, 6, 2, 2, 2, 6, 2, 5, 6, 4, 3, 3, 3, 2, 4, 3, 6, 3, 1, 4, 3, 5, 6, 5, 6,
3, 5, 6, 4, 4, 2, 6, 3, 1, 1, 6, 1, 5, 1, 6, 3, 3, 4, 4, 2, 6, 6, 2, 6, 6, 4, 5,
1, 3]
```

Let's first keep a counter for each face value and increment that counter whenever we see that face value. Start with a list of 6 zeros:

[31]:
```
import numpy as np
counters = np.zeros(6)
counters
```

[31]:
```
array([0., 0., 0., 0., 0., 0.])
```

We can use these counters to make our first plots. Let's start with a simple bar graph:

[32]:
```
for sim in range(num_sims):
    die=random.choice(range(1,7)) # range(1,7) =[1,2,3,4,5,6]
    values += [die] # values.append(die)
    counters[die-1] += 1
counters
```

```
[32]: array([165., 124., 165., 184., 181., 181.])
```

```
[33]: vals = list(range(1,7))
vals
```

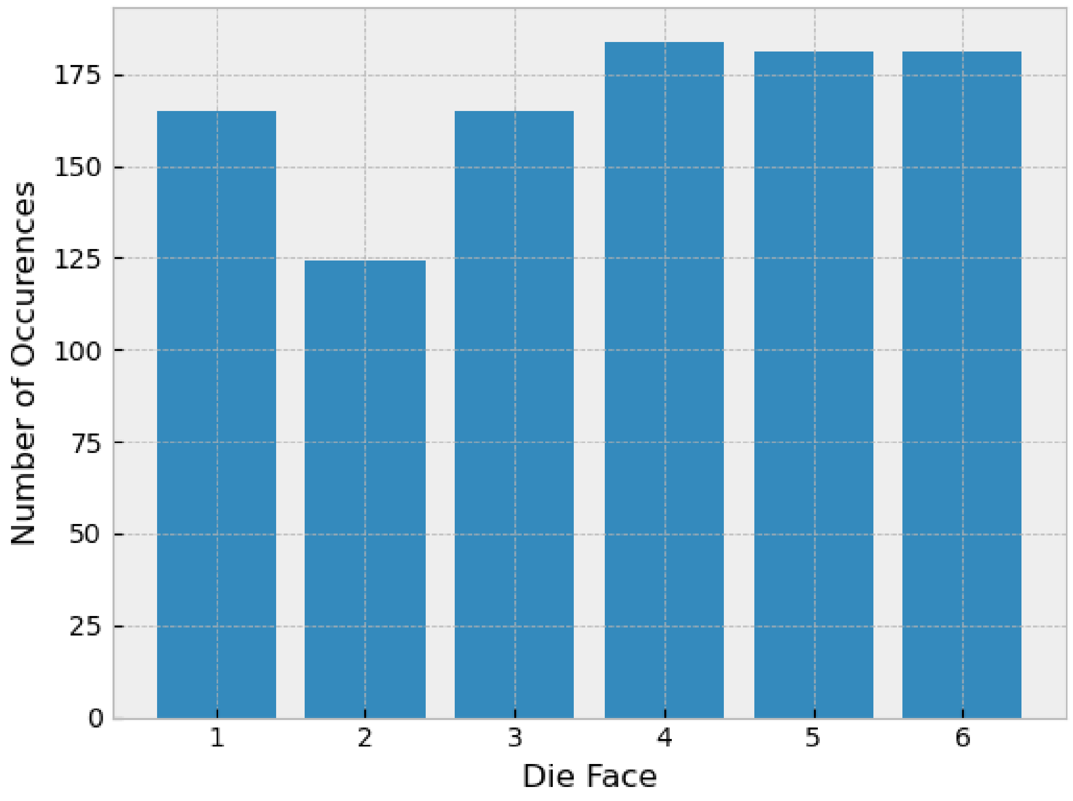
```
[33]: [1, 2, 3, 4, 5, 6]
```

Adding some labels for the bar plot

```
[34]: import matplotlib.pyplot as plt

plt.bar(vals, counters)
plt.xlabel('Die Face')
plt.ylabel('Number of Occurences')
```

```
[34]: Text(0, 0.5, 'Number of Occurences')
```



```
[ ]:
```

Here is a more elegant approach (using `numpy`) if we just want the counts of the outcomes:

[35]:
```
num_sims=100
outcomes=[]
for sim in range(num_sims):
    die=random.choice(range(1,7))
    outcomes+=[die]

# The magic counting code goes here...

# TO BE COMPLETED IN CLASS
```

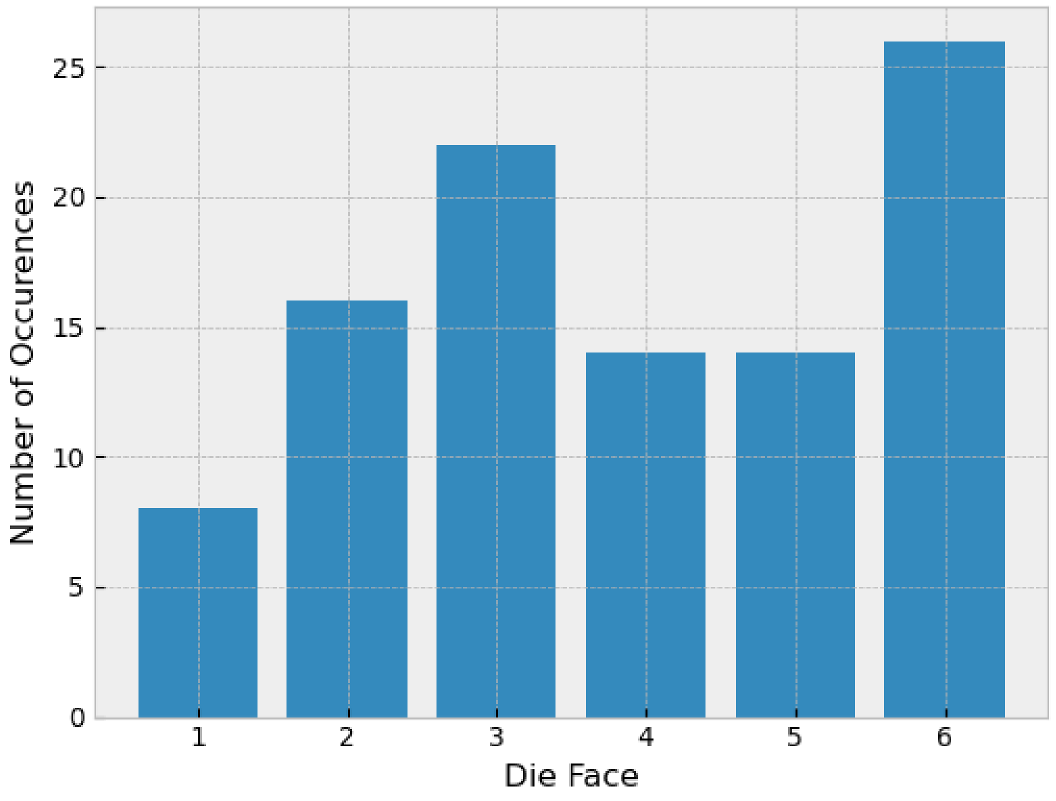
[36]:
```
np.unique(outcomes, return_counts =True)
```

[36]:
```
(array([1, 2, 3, 4, 5, 6]), array([ 8, 16, 22, 14, 14, 26]))
```

[38]:
```
vals1, counters2 = np.unique(outcomes, return_counts =True)

plt.bar(vals1, counters2)
plt.xlabel('Die Face')
plt.ylabel('Number of Occurences')
```

[38]:
```
Text(0, 0.5, 'Number of Occurences')
```



Then to get the **relative frequencies** is easy:

```
[ ]: num_sims=100
     outcomes=[]
     for sim in range(num_sims):
         die=random.choice(range(1,7))
         outcomes+=[die]

     # TO BE COMPLETED IN CLASS
```

```
[ ]:
```

- How does the relative frequency of each outcome change as we increase/decrease the number of simulations?
- What is your conclusion in terms of amount of data needed?
- Does the relative frequency *converge* to some value as the number of simulations increases?

```
[ ]:
```

```
[ ]:
```