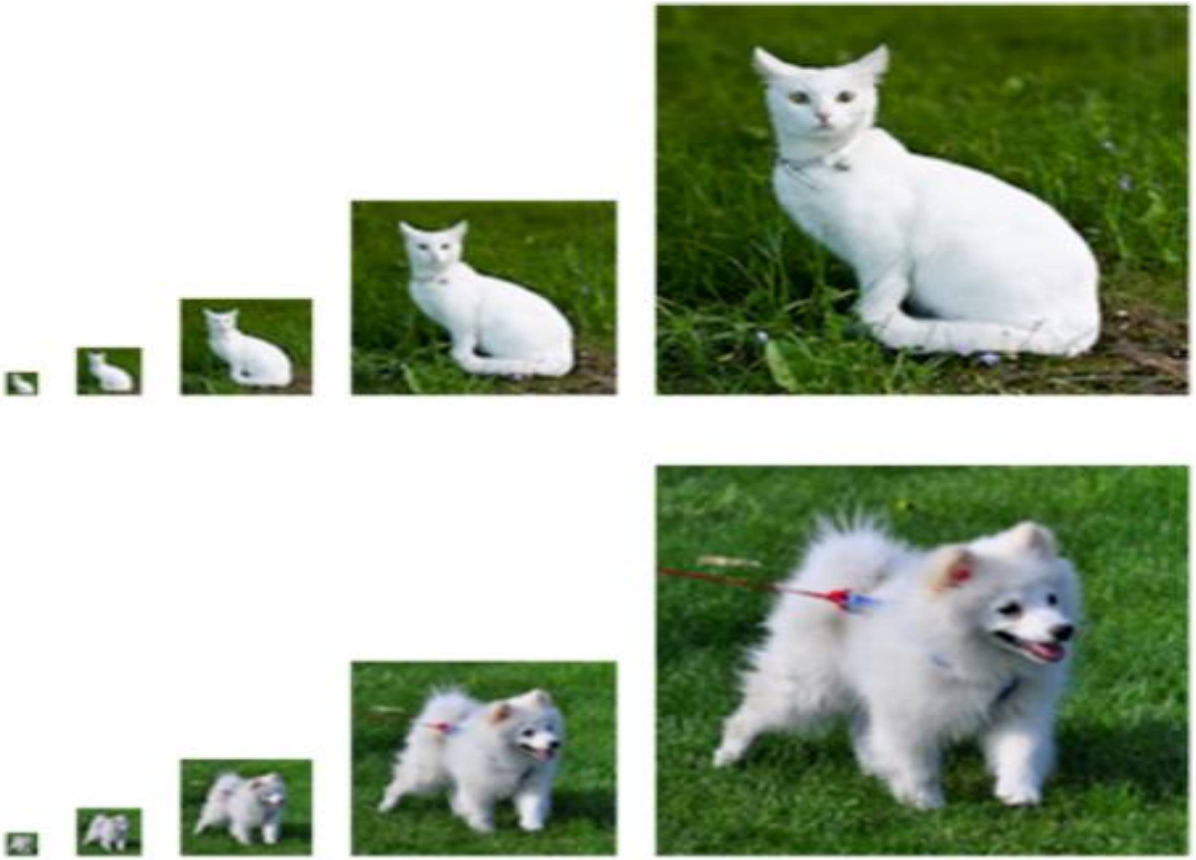
Probabilty

- Machine learning is all about making predictions.
- We might want to predict the **probability of a patient suffering a heart attack in the next year, given their clinical history.**
- Distinguishing cats and dogs based on photographs.
- This might sound simple but it is actually a formidable challenge.
- The difficulty of the problem may depend on the resolution of the image.

Probabilty…

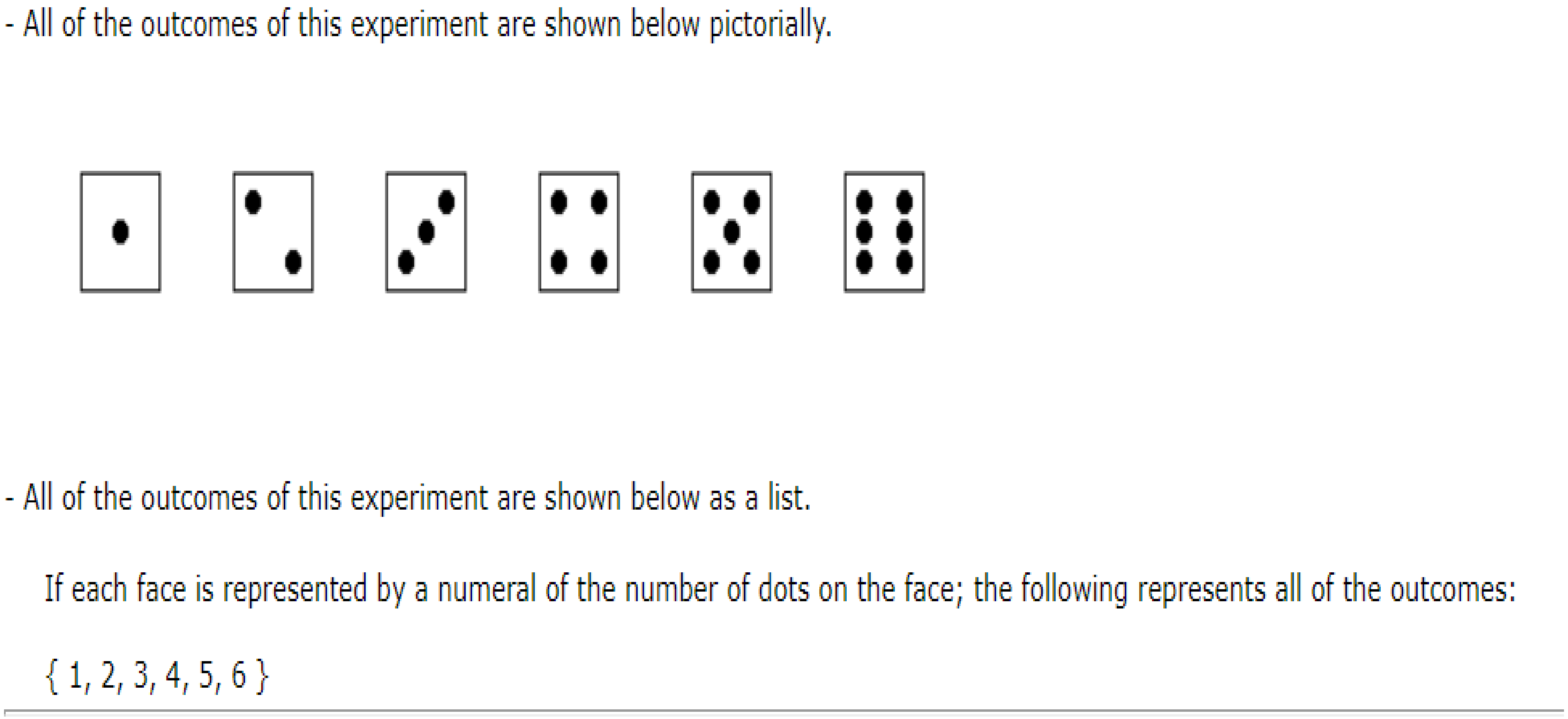


Images of varying resolutions (10 × 10, 20 × 20, 40 × 40, 80 × 80, and 160 × 160 pixels).

Probabilty…

- While it is easy for humans to recognize cats and dogs at the resolution of 160×160pixels,it becomes challenging at 40×40 pixels and next to impossible at 10×10 pixels.
- Our ability to tell cats and dogs apart at a large distance(and thus low resolution) might approach uninformed guessing. **Probability gives us a formal way of reasoning about our level of certainty.**

Write a Python program which will roll a dice and decides the probability of it.



- All of the outcomes of this experiment are shown below pictorially.

- All of the outcomes of this experiment are shown below as a list.

If each face is represented by a numeral of the number of dots on the face; the following represents all of the outcomes:

{ 1, 2, 3, 4, 5, 6 }

Roll a dice and decides the probability of it….

```
import random
 rolled = []
rolledtimes = 0;
biggest = []

freq = int(input('How many times would you like to roll the dice? '))

def roll():
    rand = random.randrange(1,7)
    return rand
def probability():
    for i in range(0,6):
        print('Calculation of probability:')
        percentage = "{:.2f}".format((count[i] / freq)*100)
        percent = str(percentage) + '%'
        print(' ', i + 1, ':', percent)
```

Roll a dice and decides the probability of it….

```
def theoretical():
    result = "{:.2f}".format((1/6)*freq)
    denominator = "{:.2f}".format(((1/6)*freq)*6)
    print('\nIn theory, each dice should roll {} out of {} times'.format(result,denominator))
def findBiggest():
for i in range(1,7):
  biggest.append(rolled.count(i))
print('\n','The most times a dice is rolled is', max(biggest),'times')
```

Roll a dice and decides the probability of it….

```
• def findSmallest():
•     for i in range(1,7):
•         biggest.append(rolled.count(i))
•     print('\n', 'The least times a dice is rolled is', min(biggest), 'times’)

 for i in range(1,freq + 1):
      number = roll()
      rolled.append(number)
     rolledtimes+=1
```

Roll a dice and decides the probability of it....

-
  ```
  count = [rolled.count(1),rolled.count(2),rolled.count(3),rolled.count(4),rolled.count(5),rolled.count(6)]
  ```

-
  ```
  print('After being rolled {} times:\n\n1 is rolled {} times\n2 is rolled {} times\n3 is rolled {} times\n4 is rolled {} times\n5 is rolled {} times\n6 is rolled {} times\n'.format(rolledtimes,count[0],count[1],count[2],count[3],count[4],count[5]))
  ```

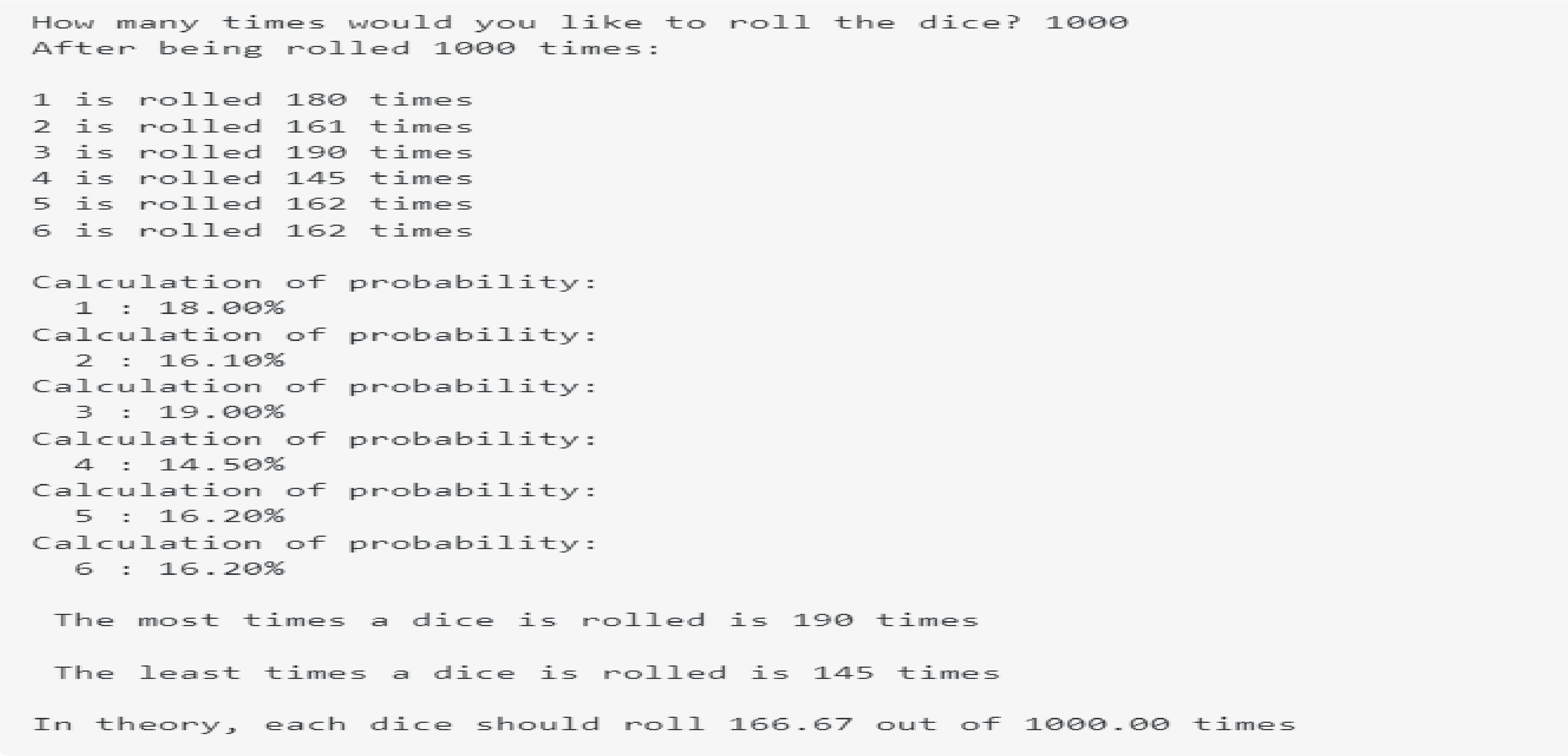
-
  ```
  probability()
  ```

-
  ```
  findBiggest()
  ```

-
  ```
  findSmallest()
  ```

-
  ```
  theoretical()
  ```

OUTPUT



```
How many times would you like to roll the dice? 1000
After being rolled 1000 times:

1 is rolled 180 times
2 is rolled 161 times
3 is rolled 190 times
4 is rolled 145 times
5 is rolled 162 times
6 is rolled 162 times

Calculation of probability:
  1 : 18.00%
Calculation of probability:
  2 : 16.10%
Calculation of probability:
  3 : 19.00%
Calculation of probability:
  4 : 14.50%
Calculation of probability:
  5 : 16.20%
Calculation of probability:
  6 : 16.20%

 The most times a dice is rolled is 190 times

 The least times a dice is rolled is 145 times

In theory, each dice should roll 166.67 out of 1000.00 times
```

Python code for finding the probability of tossing a coin

```python
import collections
import itertools
from fractions import Fraction

def fibonacci_nth(size):
    store = collections.deque([0] * size, size)
    store.append(1)
    while True:
        yield store[-1]
        store.append(sum(store))
```

Python code for finding the probability of tossing a coin

```python
def coin_chance(flips, streak):
    if streak <= 0 or streak % 1:
        raise ValueError("streak must be a positive integer")
    if flips < 0 or flips % 1:
        raise ValueError("flips must be a non-negative integer")
    if streak == 1:
        return Fraction(flips != 0, 1)
    sequence = (
        Fraction(2 * numerator, 2 ** exponent)
        for exponent, numerator in enumerate(fibonacci_nth(streak - 1), streak)
    )
    return sum(itertools.islice(sequence, flips - streak + 1))
```

Conditional Probability

Bayes’theorem :Assume that P(B)>0

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}.$$

Expectation and Variance

$$E[X] = \sum_{x} xP(X = x).$$

$$\mathrm{Var}[X] = E\left[(X - E[X])^2\right] = E[X^2] - E[X]^2.$$

Home work

- We conducted m = 500 groups of experiments where each group draws n = 10 samples. Vary m and n. Observe and analyze the experimental results.
- Write Python code for finding the probability of tossing a coin.