1. In the following problems, write what each code block would display if executed in a Jupyter cell. If it would display an error or create an infinite loop, write "error" or "infinite loop".

   (a)
   ```
   fruit = 'bitter melon'
   weight = 2.745
   f"The {fruit.split()[1]} weighs {weight:.1f} pounds."
   ```

   (b)
   ```
   n = 1
   count = 0
   while count < 4:
       print(n, end=':')
       if n % 2 == 0:
           n += 1
       else:
           n *= 2
       count += 1
   ```

   (c)
   ```
   chars = ['a', 'b']
   for i in range(1, 3):
       for j in chars:
           print(j * i, end=',')
       print()
   ```

   (d)
   ```
   def func1(nums):
       if len(nums) == 0:
           return 0
       else:
           print(nums)
       return nums[0] - func1(nums[1:])
   func1([3,4,5])
   ```

   Solution:

   (a) `'The melon weighs 2.7 pounds.'`

   (b)
   ```
   1:2:3:6:
   ```

   (c)
   ```
   a,b,
   aa,bb,
   ```

   (d)
   ```
   [3, 4, 5]
   [4, 5]
   [5]
   4
   ```

2. When decoding a coded message, a helpful thing to know is how frequently certain letters occur in a written language. Suppose that the file `article.txt` has the text from a long article, all in lowercase on a single line and with no punctuation, like:

```
there are several hundred cultivars of mango worldwide depending on the...
```

Write code which reads in the text of `article.txt`, and removes all spaces from it. Then create a dictionary with letters as keys, and their frequency in the text as values.

This article is extremely long, so your code should be as efficient as possible (meaning, do not use for-loops).

Solution:

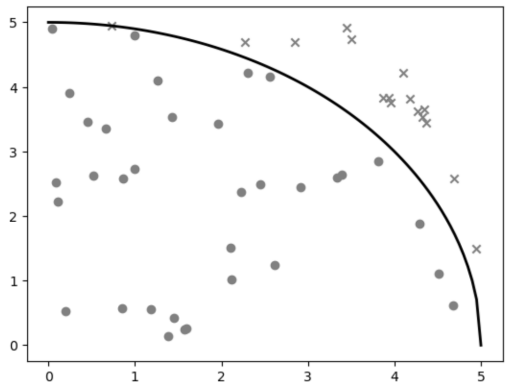
```
with open('article.txt') as f:
    text = f.read().replace(" ", "")

freq = {x: text.count(x) for x in set(text)}
```

3. Write a function `target(r, n)` which doesn't return anything, but which

   - plots the portion in the first quadrant of a circle of radius `r` centered at the origin,
   - plots `n` random points in the first quadrant whose x, y coordinates are less than or equal to `r`,
   - if a random point is within the circle, its marker should be a dot; if a random point is outside the circle, its marker should be an "x".

   The circle color should be black, but the random points don't need to be specific colors.

   For example, `target(5, 50)` might show something like



Solution:

```
def target(r, n):
    xvals = np.linspace(0, r, 100)
    yvals = [math.sqrt(r ** 2 - x ** 2) for x in xvals]
    plt.plot(xvals, yvals, color='black', lw=2)

    rpx = [r * random.random() for i in range(n)]
    rpy = [r * random.random() for i in range(n)]

    for x, y in zip(rpx, rpy):
```

```
        if (x ** 2 + y ** 2 <= r ** 2):
            plt.scatter(x, y, color='grey', marker = "o")
        else:
            plt.scatter(x, y, color='grey', marker = "x")

    plt.show()
```

4. Define a function `letter_game(word)` which randomly selects a lower-case letter one at a time until it has correctly guessed all letters in the string `word`. The function should return the total number of guesses, and it should guess the first letter in `word` before moving on to the second letter, etc.

   For example, `letter_game('ab')` might take 12 guesses before correctly guessing `'a'`, followed by 8 guesses to guess `'b'`, at which point the function would return $12 + 8 = 20$ total guesses.

   You may fill in code here so that you don't have to write out the full alphabet:

```
def letter_game(word):
    alpha = 'abcdefghijklmnopqrstuvwxyz'
```

   Solution:

```
def letter_game(word):
    alpha = 'abcdefghijklmnopqrstuvwxyz'
    guess = ''
    count = 0
    while len(guess) < len(word):
        ltr = random.choice(alpha)
        count += 1
        print(guess, end=' ')
        if ltr == word[len(guess)]:
            guess += ltr

    return count
```