1. Given the following code, answer the questions below:

```
1  def calculateExponential(base, exponent):
2    '''calculates the value of base to the power of exponent'''
3    output = 1
4    for i in range(exponent):
5      output = output * base
6
7    return output
8
9  def printExponentList(numList):
10   '''prints each number in a list to the power of its index'''
11   for i in range(len(numList)):
12     print(f"{numList[i]} to the power of {i} is {calculateExponential(numList[i], i)}")
13
14   #Add extra line of space after all numbers in the list are printed
15   print()
16
17 # Main program
18 list1 = [2, 2, 2, 2, 2, 2, 2, 2]
19 printExponentList(list1)
20
```

(a) In your own words, what does this program do? What is printed to the terminal?

This program defines 3 functions: calculate Exponential, print Exponential List, The printExponent list calls the calculate Exponential function, Then a list is defined as list1, which then gets printed as each number being printed to the power of its index.

(b) Of the above, which lines are function definitions, and which lines include function calls (to functions defined in the code snippet)? (Write the line numbers)

1, 9, 12, 19

(c) How many times is calculateExponential called throughout the duration of the program execution?

I believe its called for each # in the list, so 8 times or only once.

2. Consider the following code, and answer the questions below:

```
1 def someFunction(text, num):
2   for i in range(num):
3     print(text * i)
4
5 # Main Program
6 star = '*'
7 number = 5
8 someFunction(star, number)
```

   (a) In this code, is the variable *star* used as a parameter, or an argument? Is *text* an argument or a parameter?

   The variable star is used as an argument, text is a parameter within the function.

   (b) What is the difference between parameters and arguments?

   Parameters are variables listed in the parathses of a function defintion

   Arguments are values sent to the function when it gets called.

3. Write the following Big O complexities in order, from smallest to largest.
   $\mathcal{O}(1), \mathcal{O}(n^3), \mathcal{O}(\log(n)), \mathcal{O}(n), \mathcal{O}(3^n), \mathcal{O}(n \cdot \log(n))$

   O(1), O(log(n)), O(n), O(n log(n)), O(n3), O(3n)

4. For each of the following, select all of the equivalent Big O complexities (there may be more than one equivalent term per line). Recall the simplification rules we learned from last week: all smaller order terms and constant factors can be removed without impacting the final result.

   (a) $\mathcal{O}(1)$
   A. $\mathcal{O}(1000)$ B. $\mathcal{O}(n \cdot \frac{1}{n})$ C. $\mathcal{O}(33n^3 + n^2 + 1050)$ D. $\mathcal{O}(33n^3 + n^2)$

   (b) $\mathcal{O}(n)$
   A. $\mathcal{O}(105n + 105)$ B. $\mathcal{O}(\frac{n}{n^2})$ C. $\mathcal{O}(\log(n) + n)$ D. $\mathcal{O}(n \cdot \log(n))$

   (c) $\mathcal{O}(2^n)$
   A. $\mathcal{O}(n^2)$ B. $\mathcal{O}(2n)$ C. $\mathcal{O}(2^n + 2^{n+1})$ D. $\mathcal{O}(2^n + \log(n))$ E. $\mathcal{O}(2^{2n})$

5. Determine the Big O behavior for each of the following functions. Assume n is the length of the list parameter, and remember to apply simplification rules whenever possible.

(a)
```
def combine(lst):
    combine = ''
    for item in lst:
      combine += item
    return combine
```

Combine (lst)

(b)
```
def combine(lst):
    if len(list) % 2 == 0:
      return "even"
    else:
      i = 0
      sum = 0
      while i < len(list) ** 2:
        ind = int(i ** 0.5)
        sum += list[i]
        i += 1
      return sum
```

Combine (lst)

(c)
```
def print_nums(lst):
    if len(list) < 100:
      for item in list:
        print(item)
    else:
      for i in range(10000):
        print(i)
```

Print_nums (lst)

6. Consider the following code:

```
# Read in words from the user, and 'clean' it for later use
noun = input("Please enter a (non-proper) noun: ")
noun = noun.lower() # Make lowercase
noun = noun.strip() # Get rid of extra whitespace

adj = input("Please enter an adjective:")
adj = adj.lower() # Make lowercase
adj = adj.strip() # Get rid of extra whitespace

verb = input("Please enter a verb:")
verb = verb.lower() # Make lowercase
verb = verb.strip() # Get rid of extra whitespace

adv = input("Please enter an adverb:")
adv = adv.lower() # Make lowercase
adv = adv.strip() # Get rid of extra whitespace

print(f"The {noun} was a very {adj} {noun}. It always {adv} {verb} everywhere.")

```

   (a) As we can see, it is very repetitive. What is a function you could write to simplify the code?

   (b) What are some benefits of putting the code into a function, instead of leaving it as is?

7. At the beginning of the course, we talked about the importance of naming variables well. Do the same concepts apply to functions? Explain why or why not.

Yes. Better functions naming makes code easier to read and modify if needed.

8. A number is considered *prime* if it is divisible by only one and itself. Write pseudocode for a function that determines if a positive integer is prime. The function should return true if its integer argument is prime, and false otherwise.

   Consider how you could achieve a complexity that is strictly faster than $\mathcal{O}(n)$.

   Hint: this will be useful come studio next class.