Solutions and explanations for these problems can be found at the end of this document.

Problems:

1. Write the function **is_even_or_positive(n)** that takes in an integer n and returns True if the integer is even or positive (i.e. greater than 0), and returns False otherwise.
2. Write the function **is_factor(n, f)** that takes in two integers and returns True if f is a factor of n and False otherwise.
   Hint: use the mod operator!
3. Write the function **total_bill** that takes in two required parameters and one optional parameter. The two required parameters are **num_people** and **total** that represent the number of people present and the total pre-tip bill amount. The optional parameter is **tip** that is represented as a percent (i.e. 0.15, 0.2, etc.). If there are more than six people, then a 20% tip is applied regardless of whether an alternate tip percentage was passed in. If a specific tip isn't passed in, the tip also defaults to 20%. The function should calculate the total amount due (pre-tip total plus however much the tip is).
4. Write the function **is_it_hot(temp)** that takes in a Celsius temperature, converts it to Fahrenheit, and returns the following messages depending on the situation:

   - "It's very hot" if the temperature (in Fahrenheit) is 90 or above
   - "It's kind of hot" if the temperature is between 75 and 90 (non-inclusive)
   - "It's warm" if the temperature is in between 60 and 75 (non-inclusive)
   - "It's cold" if the temperature is below 60

   As a reminder, the formula to convert Celsius to Fahrenheit is $\frac{9}{5}C + 32$
5. Write the function **binomial_coefficient(n, k)** that calculates n choose k. As a reminder, n choose k is equal to $\frac{n!}{k!(n-k)!}$. You should use the factorial function in the math library.
6. Write the function **set_kth_digit(n, k, d)** that takes in an integer n and replaces the kth digit in n with digit d. We define the kth digit reading the number from right to left and counting starting from 0. So for example, if we had n = 2563, the 0th digit would be 3, the 1st digit would be 6, etc.

   Note this function is a continuation of **get_kth_digit(n, k)** that was a spicy problem from homework 1. You may assume that **get_kth_digit** is already written and use it as a helper function (hint hint, you should do this :)).

   You may not use strings.

Solutions:

is_even_or_positive: For this function, we need to check two things: if the integer is even and if it's positive. Checking if it's positive is straightforward – we just need to use > 0. To check if it's even, we can use %. If the number is even, then dividing it by 2 will leave no remainder. On the other hand, if the number is odd, then dividing it by 2 will not leave a remainder of 0. We can use this fact to do our evenness check.

When writing the function, there are actually many versions with different combinations of the number of if/else statements we use. Below, I'll show you three versions, but there are other variations.

Version 1 of solution: two if statements

```
def is_even_or_positive(n):
    if n % 2 == 0:
        return True
    elif n > 0:
        return True
    else:
        return False
```

In this version, we have two separate if statements to check our conditions. We chained together everything using an **if/elif/else** statement, but since each of these have return statements, it wasn't strictly necessary.

Version 2: one if statement

```
def is_even_or_positive(n):
    if (n % 2 == 0) or (n > 0):
        return True
    else:
        return False
```

Here, we used the logical operator **or** to combine both conditions into one line.

Version 3: no if statements

```
def is_even_or_positive(n):
    return (n % 2 == 0) or (n > 0)
```

This is similar to the example in the notes, where we can return the result directly because we want a boolean anyway.

All three of these versions are perfectly valid. Which version you prefer may change as you learn more!

is_factor: To determine if **f** is a factor of **n**, we just need to check if **f** divides **n** evenly. In other words, if there is a remainder when you divide **n** by **f**. Helpfully, we have a mod operator that gives us the remainder when you divide two numbers. So, we can just check if **n % f** is 0 or not.

As before, there are two versions of this function that you could write.

Version 1: one if statement

```
def is_factor(n, f):
    if n % f == 0:
        return True
    else:
        return False
```

Version 2: no if statements

```
def is_factor(n, f):
    return (n % f == 0)
```

total_bill: This one has a lot of moving parts. Let's start with the function definition. It tells us that we have two required parameters and one optional parameter, so our definition will look something like this:

```
def total_bill(num_people, total, tip = ___):
```

As we see later on in the problem statement, the tip value is defaulted to 20%, so our full definition is:

```
def total_bill(num_people, total, tip = 0.2):
```

For the body of the function, we need to calculate how much extra the tip is and add it to the total. We need to add a conditional for whether the number of people is greater than 6, in which case we need to use 0.2 as the tip rather than the **tip** variable.

```
def total_bill(num_people, total, tip = 0.2):
    if num_people > 6:
        return 1.2*total
    else:
        return (1+tip)*total
```

is_it_hot: This problem has two components: a conversion from Celsius to Fahrenheit and then a bunch of conditionals to return the correct statement.

For the conversion, we can just plug in the variable to the provided equation. Then we'll have a chain of **if/elif/else** statements. Note that in this case, we should use **if/elif/else** rather than multiple **if**s because all these conditions are mutually exclusive.

```
def is_it_hot(temp):
    fahrenheit_temp = temp*(9/5) + 32
    if fahrenheit_temp >= 90:
        return "It’s very hot"
    elif 75 <= fahrenheit_temp < 90:
        return "It’s kind of hot"
    elif 60 <= fahrenheit_temp < 75:
        return "It’s warm"
    else:
        return "It’s cold"
```

Note that we haven't seen this type of conditional before where we do two comparisons at a time like **`75 <= fahrenheit_temp < 90`**, but this works as you would expect it to.

You could also have said **`(fahrenheit_temp >= 75) and (fahrenheit_temp < 90)`**.
What you can't do is something like **`fahrenheit >= 75 and < 90.`**

binomial_coefficient: For this function, we will make use of the factorial function built-in to the math library. Once we have that, the mathematical part is pretty straightforward.

There are a couple ways we can import the factorial function in – we can either import in the entire math library or just the factorial function specifically. I'll show you the code for both.

Import #1: entire math library

```
import math
def binomial_coefficient(n, k):
    return math.factorial(n)/(math.factorial(k)*math.factorial(n-k))
```

In this version, notice we have to use **math.factorial** whenever we call the function in order to tell Python that the factorial function is part of the math library.

Import #2: just the factorial function

```
from math import factorial
```

```
def binomial_coefficient(n, k):
    return factorial(n)/(factorial(k)*factorial(n-k))
```

In this version, we can call factorial like a normal function. Be careful when importing like this though – you need to make sure you haven't written your own factorial function somewhere else!

set_kth_digit: This function's algorithm is a bit tricky. We have to work exclusively with mathematical operations. Let's illustrate our algorithm with an example. Suppose n = 258 and we want to replace the 1st digit with 3. So, our final output should be 238. Using mathematical operations, what we want to do is first "zero out" the 1st digit. In other words, we want to go from 258 to 208. Then, we want to replace the 0 with 3 to get 238. To be more specific, we can "zero out" the 5 by subtracting 50 from 258. And then we can replace the 0 with a 3 by adding 30 to 258.

Okay, this works, but it's still pretty hard to see how we can generalize this. So, let's look at another example: suppose we have n = 4689 and we want to replace the 2nd digit with 1, so the final output should be 4189. Following the previous idea, we want to first subtract 600 from 4689 to get 4089 and then add back 100 to get 4189.

With these two examples, let's try to generalize this algorithm now. The number we subtract from n seems to depend on two things: what the kth digit is, and where it appears in the number. In the first example, the kth digit was 5 and we multiplied it by 10 to get 50. In the second example, the kth digit was 6 and we multiplied it by 100 to get 600. So, the power of 10 that we multiply the kth digit by is exactly equal to k itself. Then, the amount we add back is equal to digit d multiplied by the 10^k power. Let's put this in a table to better illustrate this:

n	**k**	**d**	**kth digit**	**Subtract**	**Add**
258	1	3	5	$5*10^1$	$3*10^1$
4689	2	1	6	$6*10^2$	$1*10^2$

So, let's write our function. As hinted at in the problem statement, we can assume **get_kth_digit** has been written for us, so we can use this to get our kth digit.

```
def set_kth_digit(n, k, d):
    kth_digit = get_kth_digit(n, k)
    subtract = kth_digit * (10**k)
    add = d * (10**k)
    return n - subtract + add
```