**Project Euler 31**

**An Explanation of My Solution**

**target**  = *the value which we hope to have our coins add up to.*

**min\_ind** = *the lowest index in the list* ***coins*** *that will be used to reach the target value.*

**max\_ind** = *indices lower than* ***max\_ind*** *in the list* ***coins****, will be used to reach the target value.*

**count** = *a counter which will tell us how many ways we've found to add coins up to our target.*

Note that both **target** and **max\_ind**will change each time the function is called. This is because every time we use a coin, our new **target** becomes the difference between that coin and the original target. So if the new **target** is less than the value of a coin, there's no point in trying to use that coin to achieve that new **target** value. As a result **max\_ind** has to refer to a coin with a value less than or equal to that of the **target**.

**For instance:**

If our original target was 10 pence and we use a 5 pence coin our new target would then be 5 pence because the difference between the target and the value of the coin used is 5.

Now this is where recursion happens! We have a new target value that we need to address. In the previous example, that new target was 5. So if we continue down this rabbit hole we can imagine using a 2 pence coin to get even closer to our target of 5. In which case we would yield a new target of 3 because the difference between the 2 pence coin and our target of 5 pence is 3.

We'll notice here that if our new target is 3 pence, there's no point in trying to use a 5 pence coin to get to this new target since we would over shoot it. So instead, when we address this new target, we will use either a 2 pence coin or a 1 pence coin to achieve our target of 3. We can then continue in this manner getting lower and lower targets and using lower and lower valued coins to reach them. We'll do this until we reach zero. In this case let’s say we use another 2 pence coin and finally a 1 pence coin to top it off.

So once we reach zero we know that we've found a way to make our coins add up to the original target of 10 pence! So you'll notice in the code that there is an **if** statement asking whether the new target is **==** to 0. And in the case that this statement is True, we can add 1 to our counter signifying that we've made our coins add up to the original target.

So if we looked at each step in the algorithm it might look something like this:

*#I like to think of recursion in terms of depth. Every time you have to #call the function inside of itself, I tell myself that I’ve gone one #step deeper. So a lot of times when I’m #trying to debug something with #recursion, I’ll have the program print the depth it’s at. not #surprisingly I’ve labeled this value “Depth”, both here, and in my #code. In this particular case, we have to go deeper each time we fail #to reach to target.*

**Depth: 0**

**Primary Target = 10**

**Starting Coin: 5**

**Number of Coins Used: 1**

**Difference = 10 – 5 = 5**

**New Target = 5**

**Target Reached? No**

*#then we get into recursion*

**Depth: 1**

**Target = 5**

**Coin: 2**

**Number of Coins Used: 2**

**Difference = 5 – 2\*2 = 1**

**Target Reached? No**

*#and then one more step of recursion*

**Depth: 2**

**Target = 1**

**Coin: 1**

**Number of Coins Used: 1**

**Difference = 1 – 1 = 0**

**Target Reached? Yes**

This is a general description of the solution!