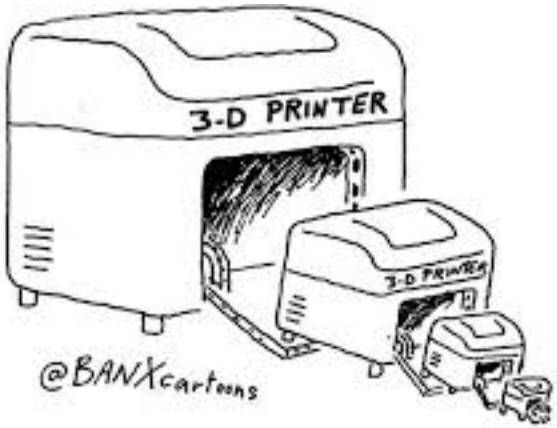
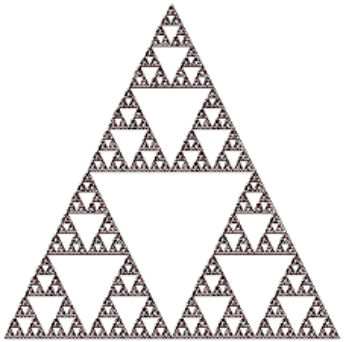
Welcome back to CS 110!

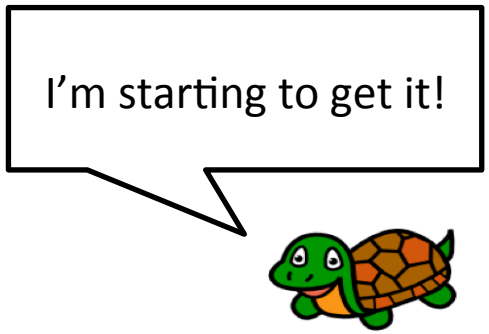
Today:

- More on Recursion

```
(factorial 6)
(* 6 (factorial 5))
(* 6 (* 5 (factorial 4)))
(* 6 (* 5 (* 4 (factorial 3))))
(* 6 (* 5 (* 4 (* 3 (factorial 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (factorial 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720
```







Let's start with an exercise

FancyFoodsMrkt has a list of products sell-by-date's (mm,dd,yy) and wants to remove all the ones that have expired to avoid getting customers sick.

```
Let's divide the problem into three steps.
1. What happens to each element?
2. What happens to the base case?
3. How is the inductive step?
```

2 and 3 are the two steps of the recursion

Clean up expired items

1. What happens to each element?

we need to check the date against today to see if its sell-by-date is older or not.

2. What happens to the base case?

 if the list is empty…

3. How is the inductive step?

 if the list is not empty…

Cleaning up expired items

1. What happens to each element?
we need to check the date against today to see if its sell-by-date is older or not.

2. What happens to the base case?
 if the list is empty…

3. How is the inductive step?
 if the list is not empty…

Tuples(Cleaning up expired items)

Lists are sequences. We've done a lot with lists.
Strings are also sequences. We've also used those a lot so far.

Let's learn about a different form of sequence in Python: Tuples.

A tuple is a sequence of elements separated by commas.
(16,12,31) is a tuple
[16,12,31] is a list

I've heard of tuples.

Tuples can be compared
(16,12, 2) > (16,11,27)

Lexicographical order

() is the empty tuple
(1,2,3) is a tuple
(1,) we need a trailing comma in tuples with only one element.
Tuples, unlike lists, cannot be updated.

Although we can write tuples without parenthesis: 1,2,3
we will use parenthesis in our examples.

Cleaning up expired items

1. What happens to each element?
we need to check the date against today to see if its sell-by-date is older or not.

2. What happens to the base case?
 if the list is empty…

3. How is the inductive step?
 if the list is not empty…

We could have done all this with a for loop…
Let's look at an example where for loops are not enough.

Cleaning up expired items

Tuples can be compared
(16,12, 23) > (16,11,27)

Lexicographical order

Those look like dates!

```
def CleanUp(lst,today):
    if lst == []:          # base case
        return []
    else:                  # inductive case
        if lst[0]<today:
            return [lst[0]]+ CleanUp(lst[1:],today)
        else:
            return CleanUp(lst[1:],today)
```

```
>>> CleanUp([(15,12,13), (16,11,12), (16, 12, 31), (16, 11, 6)], (16, 11, 7))
[(15, 12, 13), (16, 11, 6)]
>>>
```

The Protein Problem

Mystery protein: Weight 12
Amino acid weights: [2, 3, 4, 7, 10, 42]

```
>> subset(12, [2, 3, 4, 7, 10, 42])
True
>>> subset(8, [2, 3, 4, 7, 10, 42])
False
>>> subset(15, [2, 3, 4, 7, 10, 42])
???
```

The Protein Problem

target list

```
>>> subset(4, [1, 2, 3, 5])
```

True or False?

We won't answer that question, but we will answer subset questions with smaller targets and/or shorter lists !

What about
`subset(10, [8, 3, 7])`

Don't be greedy!

Writing subset

```
>>> subset(4, [1, 2, 3, 5])
True

def subset( target , L ):
    if target == 0: return ???
```

Look! A Python “one liner!”

Writing `subset`

```
>>> subset(4, [1, 2, 3, 5])
True

def subset( target , L ):
    if target == 0: return True
    elif L == ??? : return ???
```

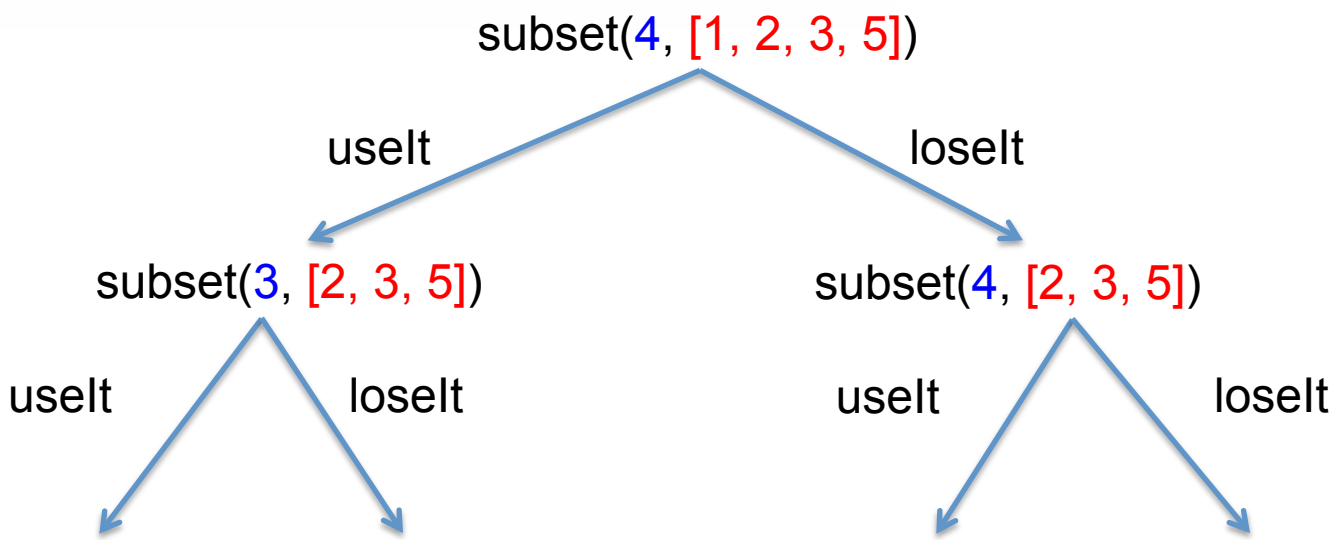
Writing subset

```
>>> subset(4, [1, 2, 3, 5])
True

def subset( target , L ):
    if target == 0: return True
    elif L == [] : return False
    elif L[0] > target: return ???
```

Writing subset

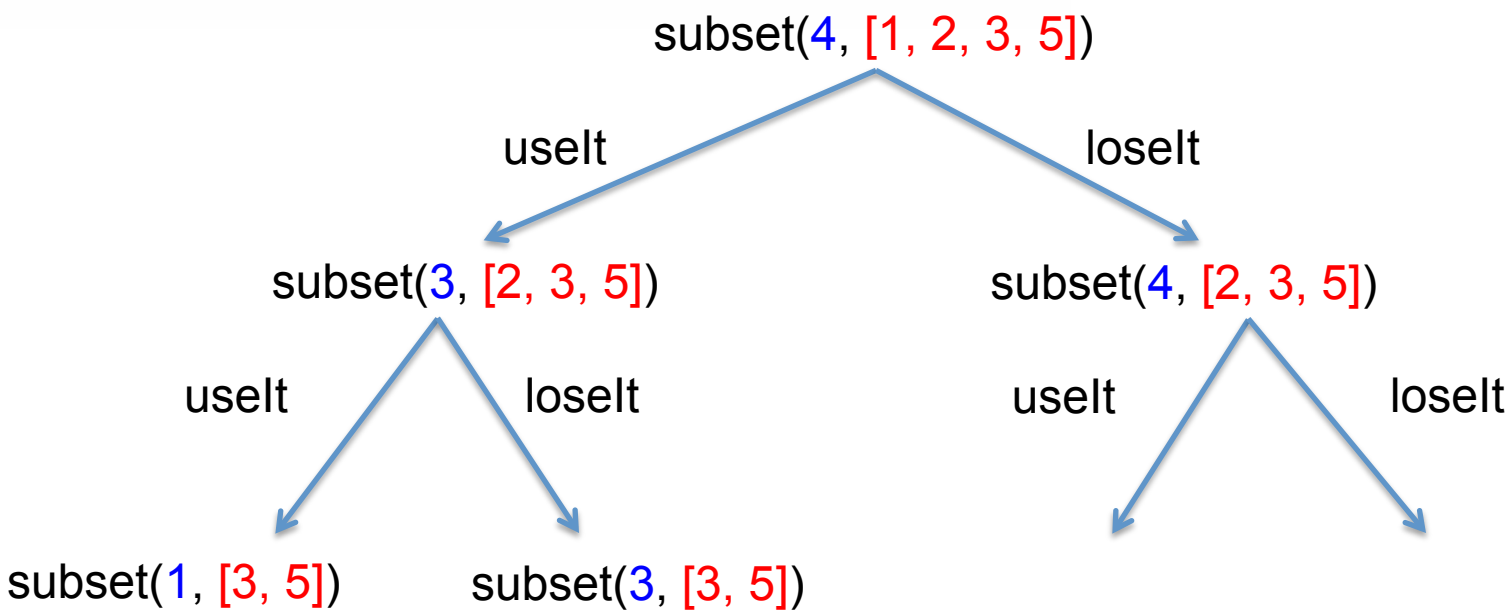
```
>>> subset(4, [1, 2, 3, 5])
True

def subset( target , L ):
    if target == 0: return True
    elif L == [] : return False
    elif L[0] > target: return subset(target, L[1:])
    else:
        It = L[0]  # aka, the "weirdo"!
        useIt =
        loseIt =
        return _____________________________
```

```python
def subset(target, L):
    if target == 0: return True
    elif L == []: return False
    elif L[0] > target: return subset(target, L[1:])
    else:
        useIt = subset(target-L[0], L[1:])
        loseIt = subset(target, L[1:])
        return useIt or loseIt
```



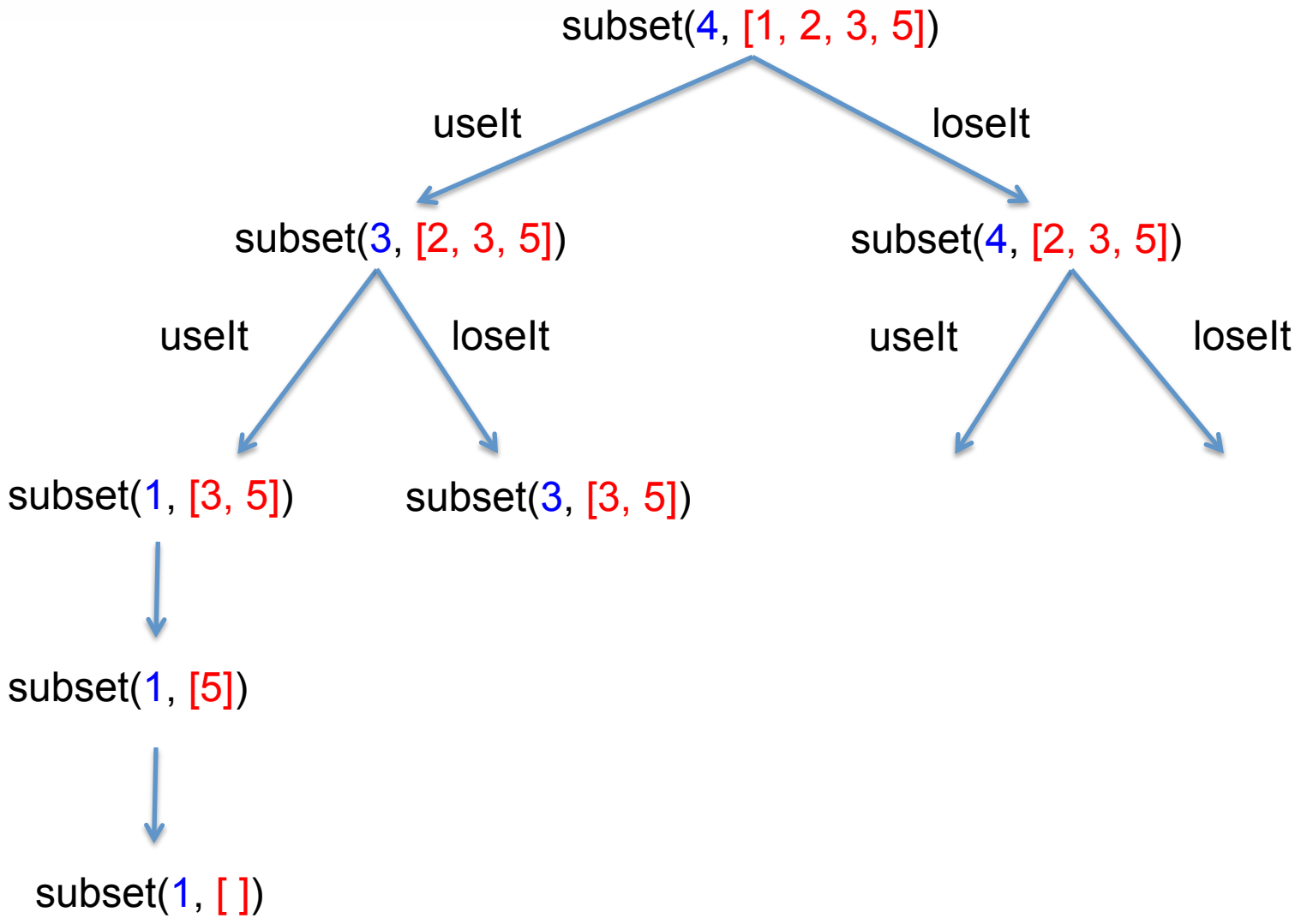
Fill this in!

```python
def subset(target, L):
    if target == 0: return True
    elif L == []: return False
    elif L[0] > target: return subset(target, L[1:])
    else:
        useIt = subset(target-L[0], L[1:])
        loseIt = subset(target, L[1:])
        return useIt or loseIt
```



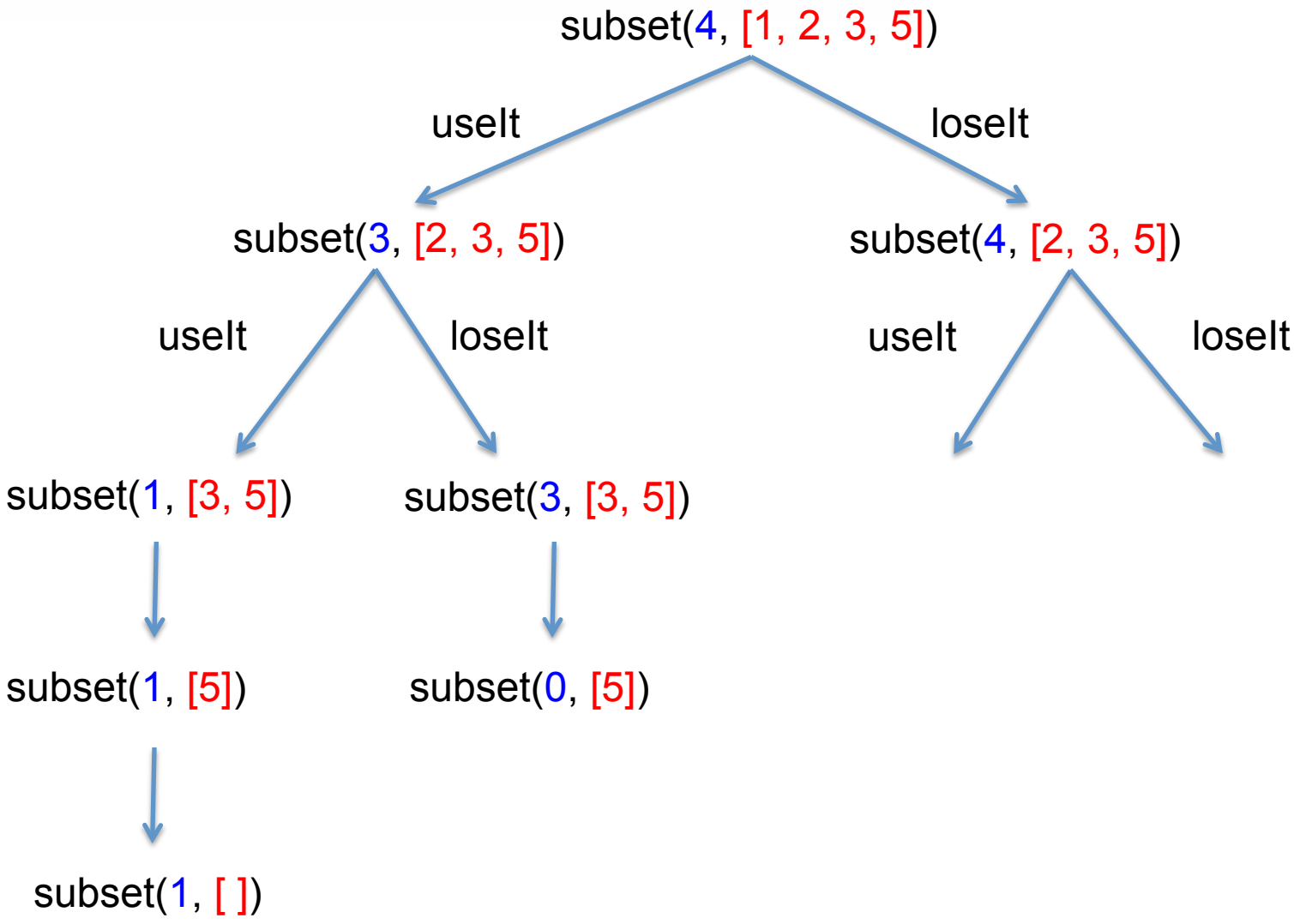
Fill this in!

```python
def subset(target, L):
    if target == 0: return True
    elif L == []: return False
    elif L[0] > target: return subset(target, L[1:])
    else:
        useIt = subset(target-L[0], L[1:])
        loseIt = subset(target, L[1:])
        return useIt or loseIt
```



Fill this in!

```
def subset(target, L):
    if target == 0: return True
    elif L == []: return False
    elif L[0] > target: return subset(target, L[1:])
    else:
        useIt = subset(target-L[0], L[1:])
        loseIt = subset(target, L[1:])
        return useIt or loseIt
```



Fill this in!

Change


```
>>> change(42, [25, 10, 5, 1])
5
>>> change(42, [10, 5, 1])
6
```

But in Shmorbodia…

Change

```
>>> change(42, [25, 10, 5, 1])
5
>>> change(42, [10, 5, 1])
7
>>> change(42, [25, 21, 1])
```

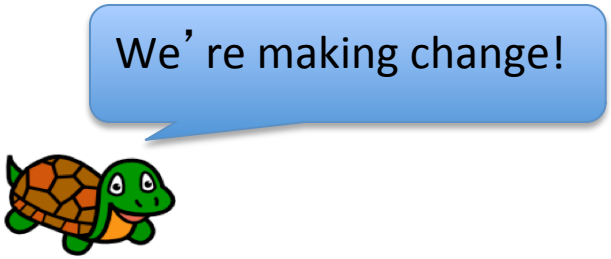
Change

```
>>> change(42, [25, 21, 1])
2
```

```
def change(amount, Coins):
```

Change

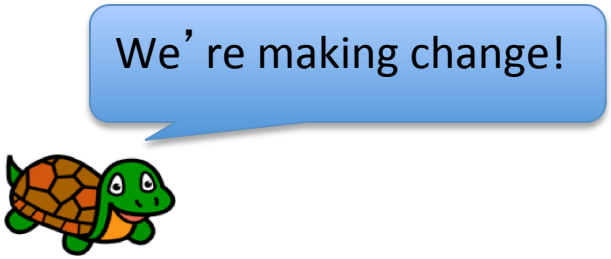
```
>>> change(42, [25, 21, 1])
2
```




```
def change(amount, Coins):
    if amount == 0: return ???
```

Change

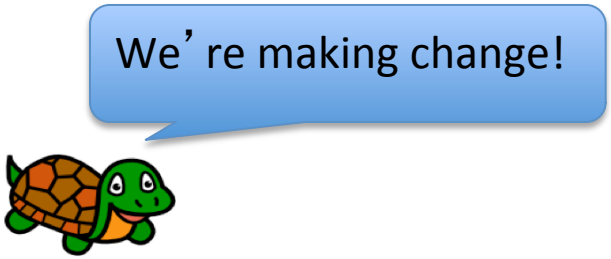
```
>>> change(42, [25, 21, 1])
2
```




```
def change(amount, Coins):
    if amount == 0: return 0
    elif Coins == []: return ???
```

Change

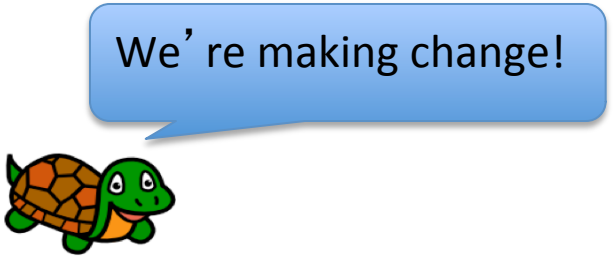
```
>>> change(42, [25, 21, 1])
2
```




```
def change(amount, Coins):
    if amount == 0: return 0
    elif Coins == []: return float('inf')
    else:
        firstCoin = Coins[0]
```

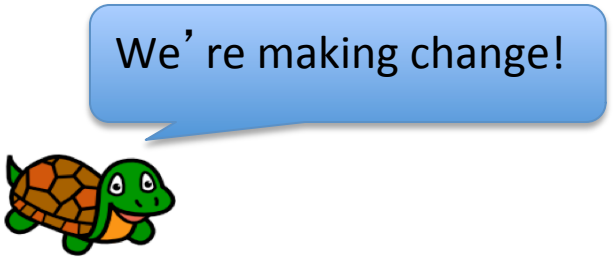
Change

```
>>> change(42, [25, 21, 1])
2

def change(amount, Coins):
    if amount == 0: return 0
    elif Coins == []: return float('inf')
    else:
        firstCoin = Coins[0]
        if firstCoin > amount: return ???
        else:
```



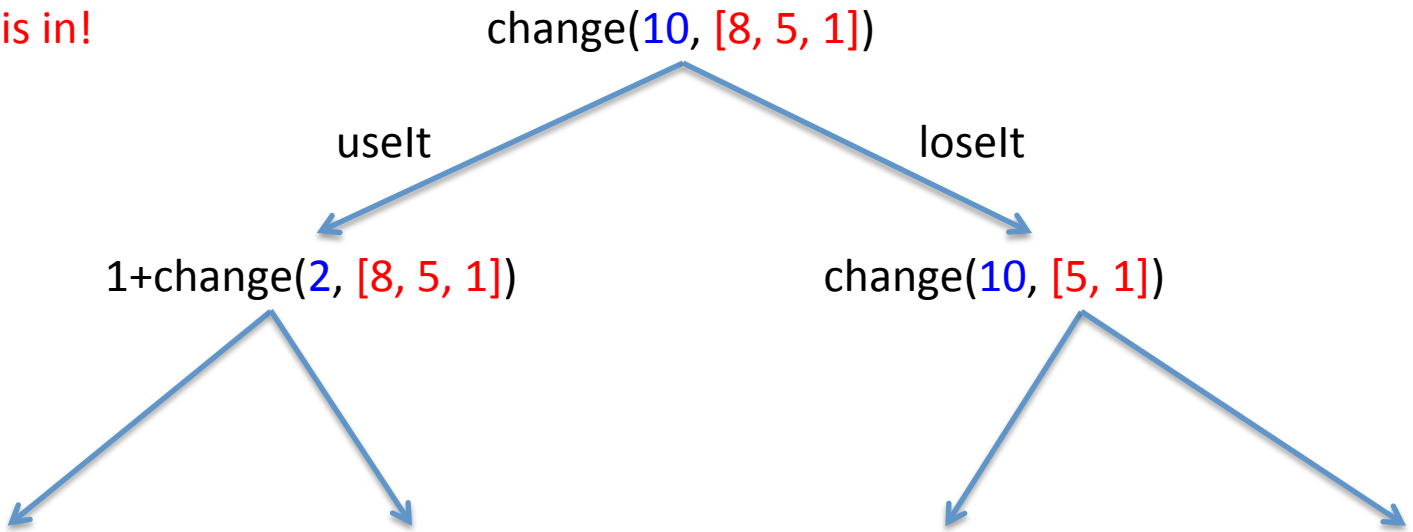
Change

```
>>> change(42, [25, 21, 1])
2

def change(amount, Coins):
    if amount == 0: return 0
    elif Coins == []: return float('inf')
    else:
        firstCoin = Coins[0]
        if firstCoin > amount: return change(amount, Coins[1:])
        else:
            useIt =

            loseIt =


            return _____________________________________________
```

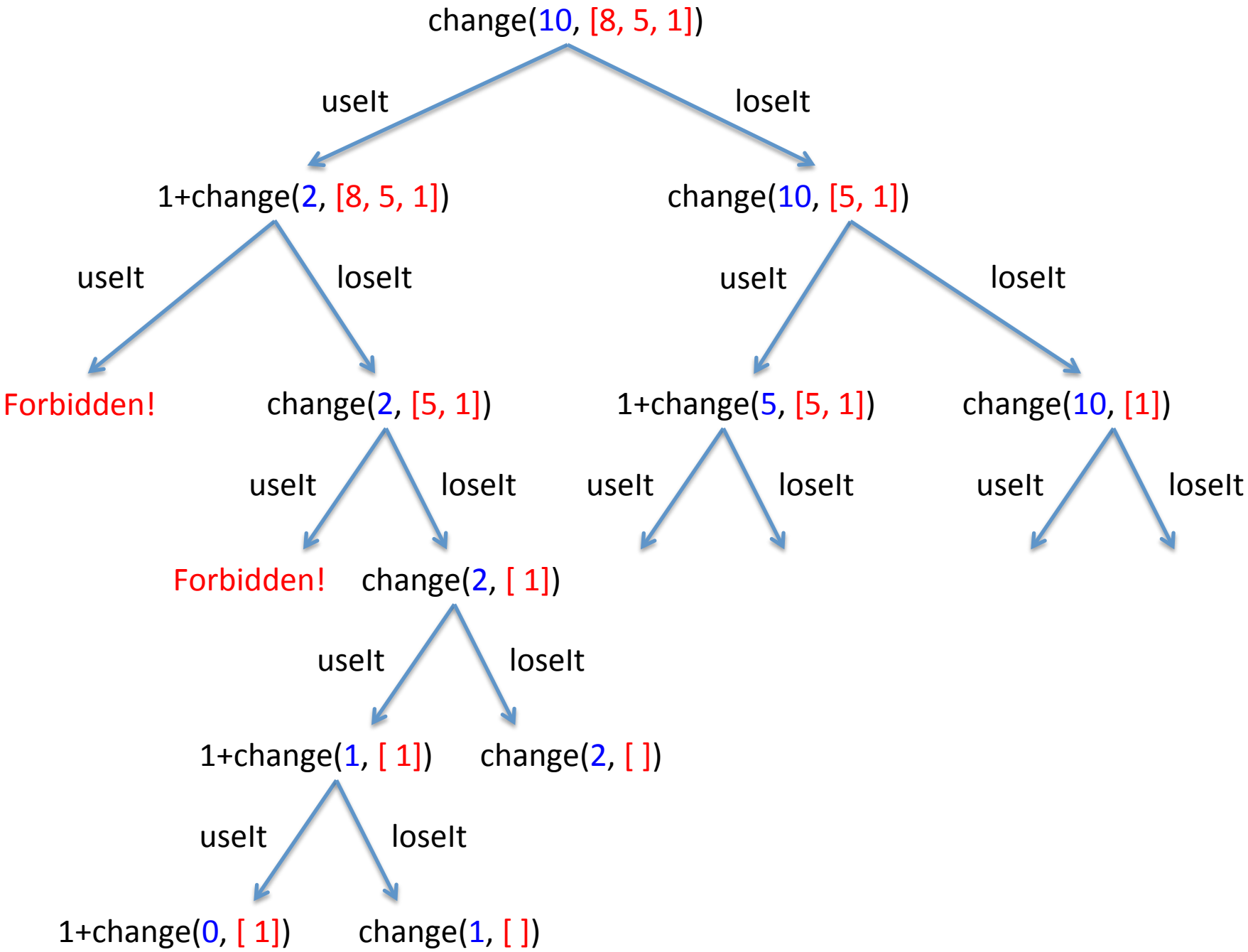


Fill this in!

Fill this in!




```
def change(amount, Coins):
    ''' returns the least number of coins required to dispense
    the given amount.'''

    if amount == 0: return 0
    elif Coins == []: return float('inf')
    else:
        firstCoin = Coins[0]
        if firstCoin > amount: return change(amount, Coins[1:])
        else:
            useIt = 1 + change(amount-firstCoin, Coins)
            loseIt = change(amount, Coins[1:])
            return min(useIt, loseIt)
```


change(10, [8, 5, 1])
useIt
loseIt
1+change(2, [8, 5, 1])
change(10, [5, 1])
useIt
loseIt
useIt
loseIt
Forbidden!
change(2, [5, 1])
1+change(5, [5, 1])
change(10, [1])
useIt
loseIt
useIt
loseIt
useIt
loseIt
Forbidden!
change(2, [1])
useIt
loseIt
1+change(1, [1])
change(2, [ ])
useIt
loseIt
1+change(0, [1])
change(1, [ ])

Comparing DNA with Longest Common Subsequence (LCS)

Groovy Groody, profdude!

AGGACAT

ATTACGAT

```
EGS = "GTACGTCGATAACTG"
WGS = "TGATCGTCATAACGT"
```

Schlitz gene


```
>>> LCS("AGGACAT", "ATTACGAT")
5
```

```
LCS("honey", "hive")
```

honey
hive

Try writing LCS

```
>>> LCS("AGGACAT", "ATTACGAT")
5
```

Now, ***you*** write LCS!

```
def LCS(String1, String2):
```

Base case(s)?
First symbols match?
Otherwise?

Edit Distance

```
>>> ED("ATTATCG", "ACATTC")
4

ATTAT-CG
A-CATTC-

>>> ED("spam", "scramble")
5
```


```
sp_am___
scramble

spam ->
scam ->
scram ->
scramb -> scrambl -> scramble
```

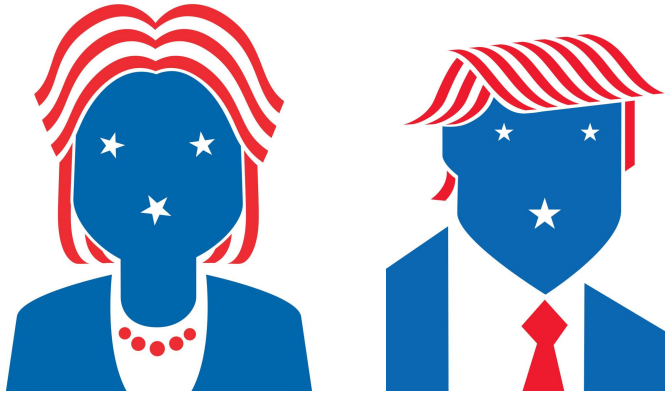
ED: Edit Distance


```
>>> ED("spam", "scramble")
5

spam ->
scam ->
scram ->
scramb -> scrambl -> scramble
```

```
def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ED(S1[1:], S2[1:])
    else:  # substitute, insert, or delete!
     substitute = 1 + ED(S1[1:], S2[1:])
     insert = 1 + ED(S1, S2[1:])
     delete = 1 + ED(S1[1:], S2)
     return min(substitute, insert, delete)
```

Election 2016

According to Real Clear Politics, the toss up states and corresponding electoral votes are:
Arizona (11), Colorado (9), Florida (29),
Georgia (16), Iowa (6), Maine (2),
Maine CD2 (1), Michigan (16), Nevada (6),
New Hampshire (4), New Mexico (5),
North Carolina (15), Ohio (18),
and Pennsylvania (20).



The Clinton/Kaine ticket needs 54 electoral votes from the toss up states while Trump/Pence needs 106 electoral votes from those states.

Write a Python function that calculates the minimum number of states that a candidate needs to win in order to secure the election.
Bonus: Modify your function so that it returns all possible combinations of states that will secure a win.