xkcd, Fixing Problems

Homework2 recap

Check the feedback that the grutors give you!

Don’t make the same mistakes twice!

We will be assuming you understand `for` and `while` loops

Please reach out if you are confused or need more help

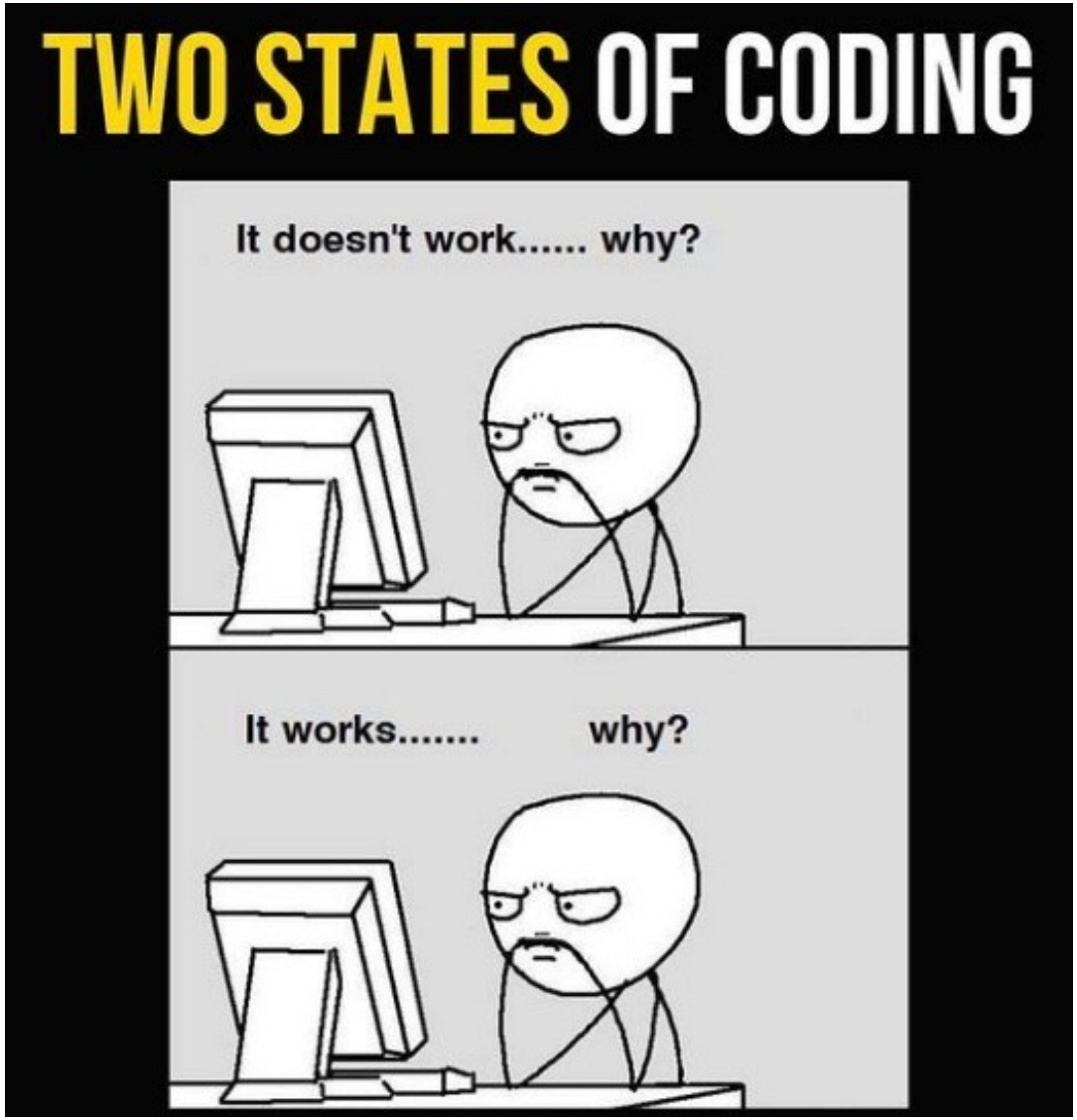
Submit all the required files!

Many people did not submit `geneFinder.txt` properly

This made the autograder award 0 points!

CHECK THE AUTOGRADER RESULTS AFTER SUBMITTING!

Maximizing the autograder's utility



Submit **incomplete** assignments!

You can **resubmit** as many times as you want before the deadline!

This will help find bugs in your functions before they cascade into bigger problems.

CS 5 Green

Learning Goals

- Review recursion
- Explain the use-it-or-lose-it strategy for recursion

Temp variables are not req'd

```
>>> sum([10, 2, 30])
42
```

There are no extra variables here, as in …
`counter = 0`

```
def sum(myList):
    """Return sum of the numbers in myList"""
    # bc: empty list => 0
    if myList == []:
        return 0

    # rc: add 1st item to remaining sum
    else:
        return myList[0] + sum(myList[1:])
```

Temp variables can be used …

```
>>> sum([10, 2, 30])
42
```


```
def sum(myList):
    """Return sum of the numbers in myList"""
    # bc: empty list => 0
    if myList == []:
        return 0

    # rc: add 1st item to remaining sum
    else: # all the variables are used and/or returned!
        firstNum = myList[0]
        remL = myList[1:]
        remSum = sum(remL)
        return firstNum + remSum
```

… but only if their values get used

```
>>> sum([10, 2, 30])
42
```


```
def sum(myList):
    """Return sum of the numbers in myList"""
    counter = 0  # counter will be reset every time
    # bc: empty list => 0
    if myList == []:
        return 0

    # bad code.
    else:
        counter += myList[0]
        sum(myList[1:]) # this value is not saved!
        return counter  # counter == myList[0]
```

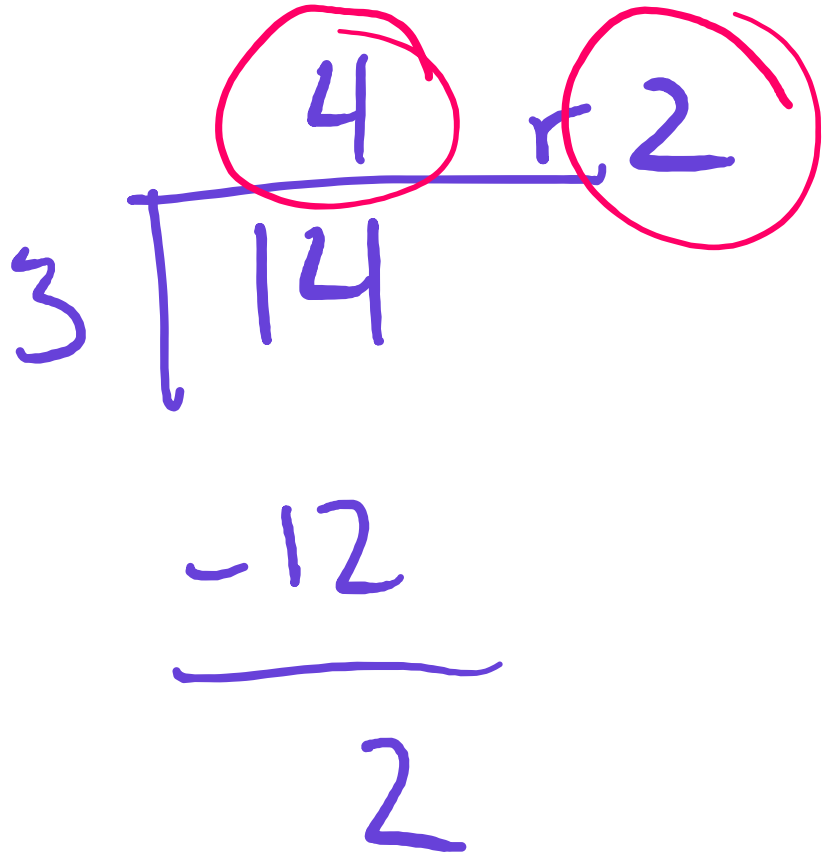
… but only if their values get used

```
>>> sum([10, 2, 30])
42
```


```
def sum(myList):
    """Return sum of the numbers in myList"""
    counter = 0  # counter will be reset every time
    # bc: empty list => 0
    if myList == []:
        return 0

    # good code.
    else:
        counter += myList[0]
        counter += sum(myList[1:]) # this works now!
        return counter
```

Quotient division and Modulo (mod)

```
>>> 14 / 3 # division
4.666666666666667

>>> 14 // 3 # quotient
4

>>> 14 % 3 # mod
2
```



Mystery!


```
def mystery(a):
    """what's a docstring? (ノಠ益ಠ)ノ彡┻━┻"""
    b = 2
    c = b // b # comments aren't important, right? (╥_╥)
    d = c - c
    if a == []: return d
    elif len(a) == c: return a[d]
    else: return mystery(a[:len(a)//b]) + mystery(a[len(a)//b:])
```

Mystery!


```
def mystery(myList):
    """what's a docstring? (ノಠ益ಠ)ノ彡┻━┻"""
    if myList == []:
        return 0

    elif len(myList) == 1:
        return myList[0]

    else:
        splitIdx = len(myList) // 2
        frontHalf = myList[:splitIdx]
        backHalf  = myList[splitIdx:]
        return mystery(frontHalf) + mystery(backHalf)
```


SPAM
Hormel
Foods

The Scale Problem

~~Gold~~ Spam nugget is claimed to weigh 12 kilos
Weights: [2, 3, 4, 7, 10, 42]

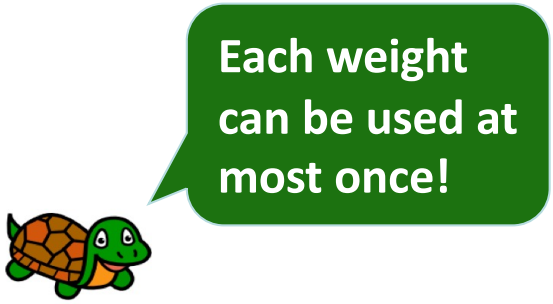
Each weight can be used at most once!

```
>> 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])
???
```

A *greedy* solution

Strategy:

find the closest value less than or equal to the target
subtract that value, and recurse with the remaining list
find the next value until we reach 0 or nothing left to pick
`True` if target is 0; `False` if nothing left to pick



The Use-it-or-lose-it Solution

Goal:

try all combinations of numbers that might make sense

Two base cases:

if our target reaches 0, then a subset exists

if our list becomes empty, then a subset does not exist

Three recursive cases:

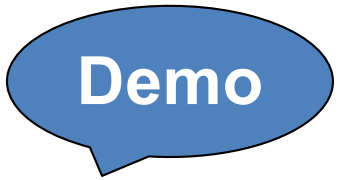
if a number exceeds our target, then we don't need it

otherwise,

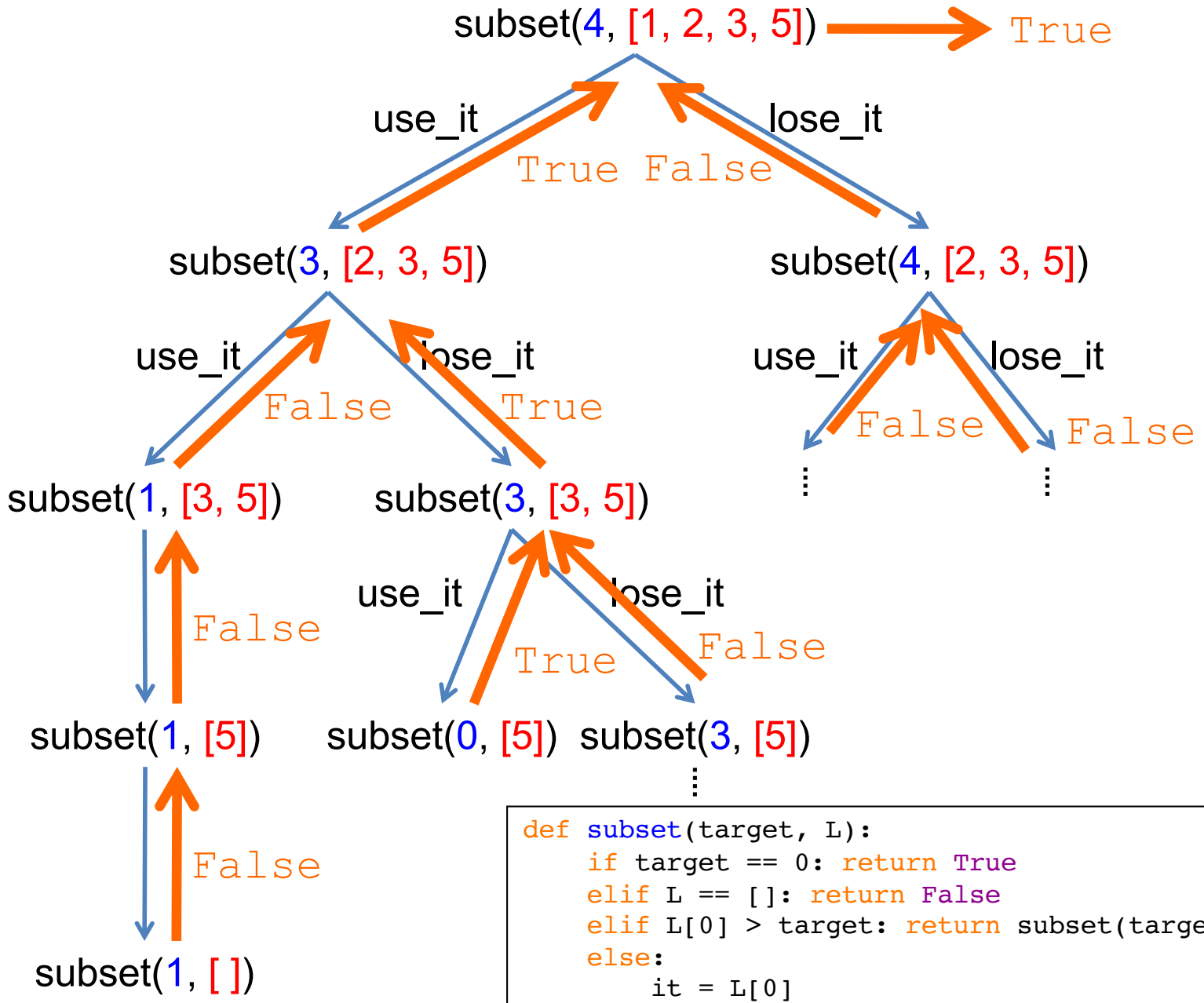
try using a number and recursing

try losing a number and recursing

Writing `subset`



Fill this in (in your notes)!




```
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]
        use_it = subset(target – it, L[1:])
        lose_it = subset(target, L[1:])
        return use_it or lose_it
```

Change

How many coins do we need?

```
>>> change(42, [25, 10, 5, 1])
5
>>> change(42, [10, 5, 1])
6
>>> change(42, [25, 21, 1])
2
```

"greedy" approach does not work!

But in Shmorbodia…

In this problem, we are allowed to use a coin denomination as many times as we want!

Change

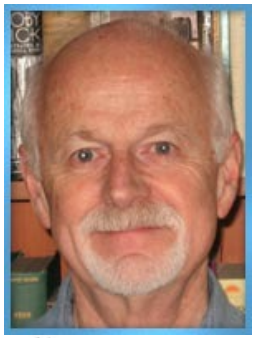
Worksheet

Q

Try to make change

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

def change(amount, coins):
    """Returns the minimum number of coins needed to
    make change for the given amount"""
    if amount == 0: return _____

    elif coins == []: return _____
    else:
        it = coins[0]
        if amount < it:
            return _____
        else:
            useIt  = _____

            loseIt = _____

            return _____
```

Comparing DNA with Longest Common Subsequence (LCS)



AGGACAT
ATTACGAT

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

Schlitz gene


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


```
LCS("spam", "pims")
```

Try writing LCS

Base case(s)?
First symbols match?
Otherwise?
`max(x, y)` is built-in

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

```
def LCS(string1, string2):
    if string1 == "" or string2 == "": return 0
    elif string1[0] == string2[0]:
        return 1 + LCS(string1[1:], string2[1:])
    else:
        return LCS(string1[1:], string2[1:])
```

```
LCS("spam", "pam") -> LCS("pam”, "am") ->
LCS("am", "m") -> LCS("a", "") -> 0
```

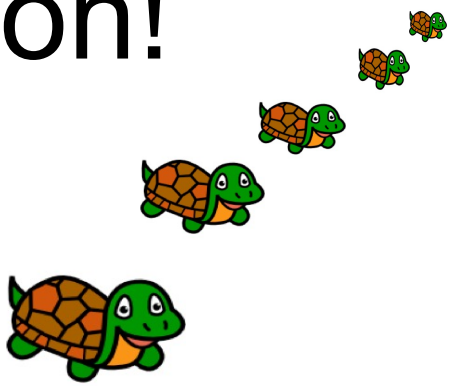
Try writing LCS


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

def LCS(string1, string2):
    if string1 == "" or string2 == "": return 0
    elif string1[0] == string2[0]:
        return 1 + LCS(string1[1:], string2[1:])
    else:
        option1 =
        option2 =
        return max(option1, option2)
```

Edit Distance (a sneak preview of things to come!)

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

ATTAT-CG
A-CATTC-


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

sp_am___
scramble

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

Turtle Meets Recursion!

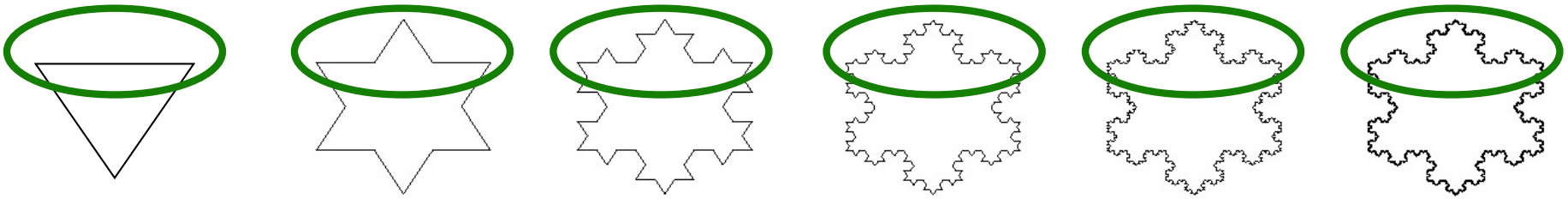


```
import turtle

def mystery(leg_length, num_legs):
    if num_legs == 0: return
    else:
        turtle.forward(leg_length)
        turtle.right(90)
        mystery(leg_length + 10, num_legs - 1)
        return
```

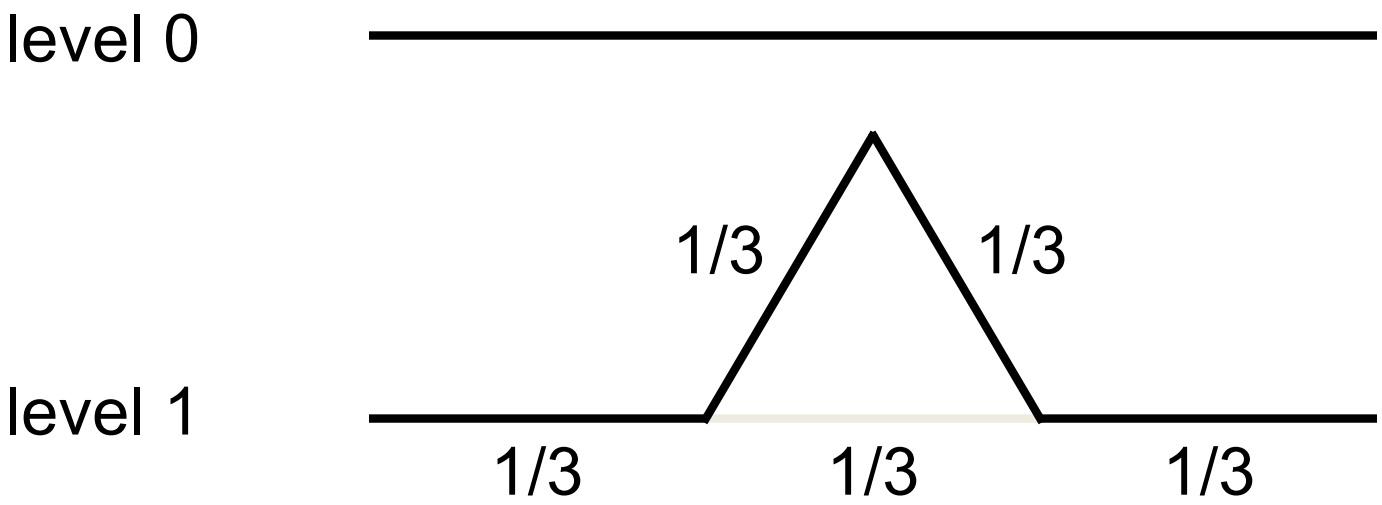
Demo

Turtle + Fractals = <(˚o˚<)

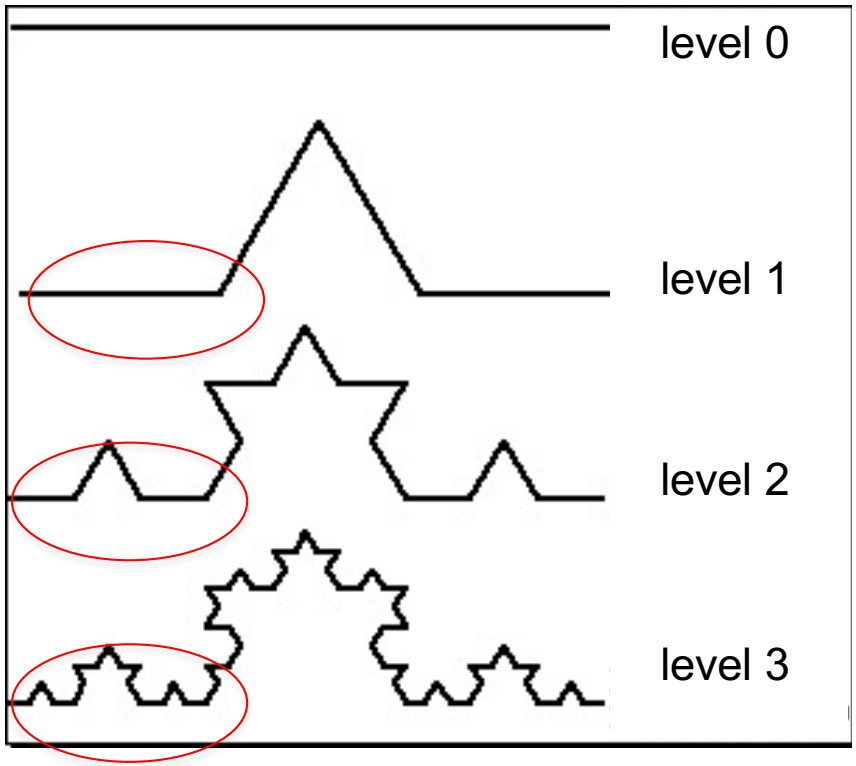
The Koch Snowflake Fractal:



level 0 level 1 level 2 level 3 level 4 level 5

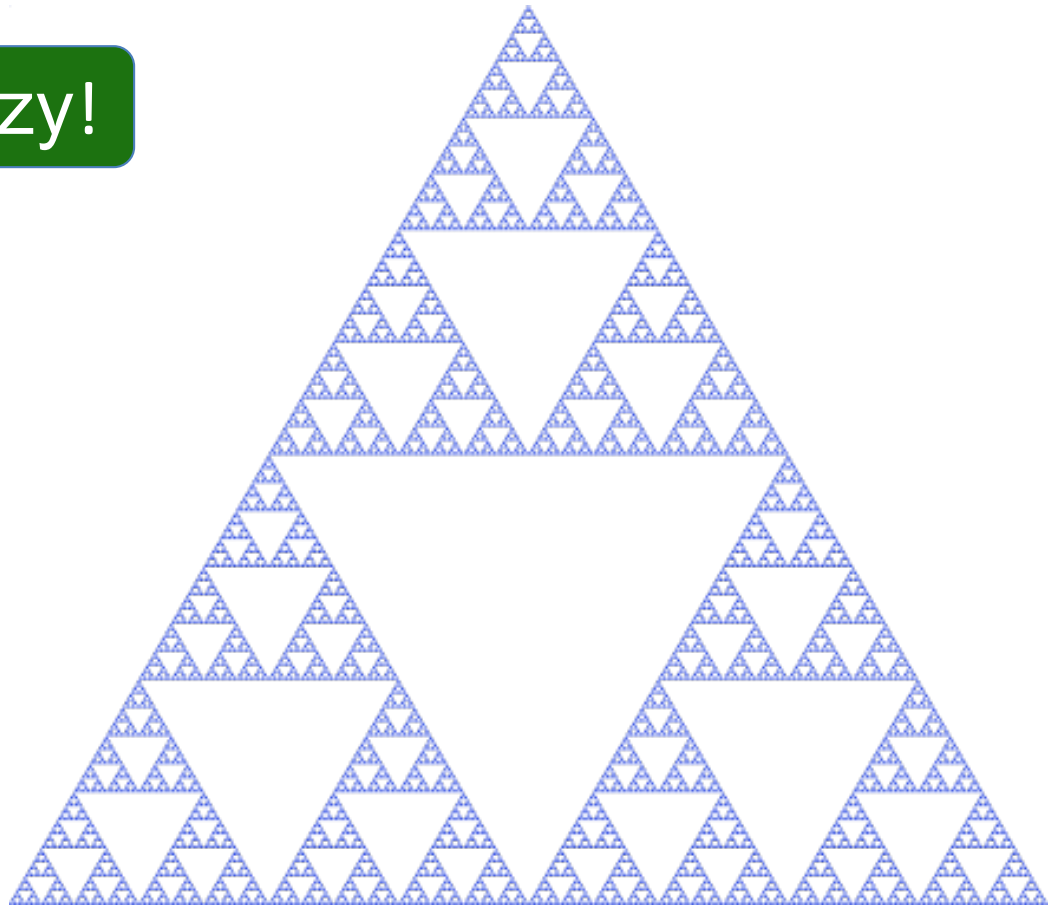


```
import turtle

def koch_side(length, level):
    if level == 0:
        turtle.forward(length)
    else:
        koch_side(length/3, level-1)
        turtle.left(60)
        koch_side(length/3, level-1)
        turtle.right(120)
        koch_side(length/3, level-1)
        turtle.left(60)
        koch_side(length/3, level-1)

def koch_flake(length, level):
    for i in range(3):
        koch_side(length, level)
        turtle.right(120)
```



Demo

The Sierpinski Triangle!



Reminder:

- Lecture feedback form (https://forms.gle/aPmkpXDUTp4Xo4CV7)