Discrete Structures!

CMPSC 102

Sets

ALLEGHENY COLLEGE

Playing With Code

Data from the “file”

```
data_text = """1972-01-01,84.700
1973-01-01,85.500
1974-01-01,86.100
1975-01-01,87.000
1976-01-01,87.600
1977-01-01,87.600
1978-01-01,88.000
"""

print(data_text)
```

Challenges When Using CSF Files?

What could possibly go wrong?!

Data from the “file”

```
data_text = """1972-01-01,84.700
1973-01-01,85.500
1974-01-01|86.100
1975-01-01;87.000
1976-01-01,
87.600
87.600;1977-01-01
1978-01-01,88.000
"""
print(data_text)
```

Higher-Order Sequence Functions

- These **Higher Order** functions should work for lists, ordered pairs, tuples:
    - **map**: Apply a function to every element of a sequence
    - **filter**: Apply a boolean function to every element of a sequence, returning only those matching the filter's rules
    - **reduce**: Apply a function that acts like a binary operator to a sequence of values, combining them to a single value
- These three operators give a **vocabulary** for implementing complex, yet easy-to-ready programs in a functional programming style
- These functions are **higher-order** because they accept function as input

Map Function with a Literal Tuple

```
def square(value):
        return value * value

def map(callFunction, sequence):
        result = ( )
        for element in sequence:
                result += ( callFunction(element), )
        return result

squared = map(square, (2, 3, 5, 7, 11))
print(squared)
```

Filtering Even Numbers from a Tuple

```
def is_even(value):
        if value % 2 == 0:
                return True
        return False

filtered_even = filter(is_even, (2, 3, 4, 5, 7, 11))
print(list(filtered_even))
```

- What does this code do?

Summations By Using Reduce

```
def plus(number_one, number_two):
        return number_one + number_two

def reduce(callFunction, sequence, initial):
        result = initial
        for value in sequence:
                result = callFunction(result, value)
        return result

numbers = [1, 2, 3, 4, 5]
added_numbers = reduce(plus, numbers, 0)
print(f"Added numbers: {added_numbers}")
```

- What does this code do?

Monoids and Map-Filter-Reduce

- **Higher-order sequence functions** are **independent** and free of side effects and thus can be **parallelized**
- Since a **monoid** has the associativity property, can use map, filter, and reduce operators in **parallel** and then combine the solution, often achieving a **speedup**. This makes the program more efficient!

Key Questions and Learning Objectives

- How do I use the mathematical concepts of sets and Boolean logic to design Python programs that are easier to implement and understand?

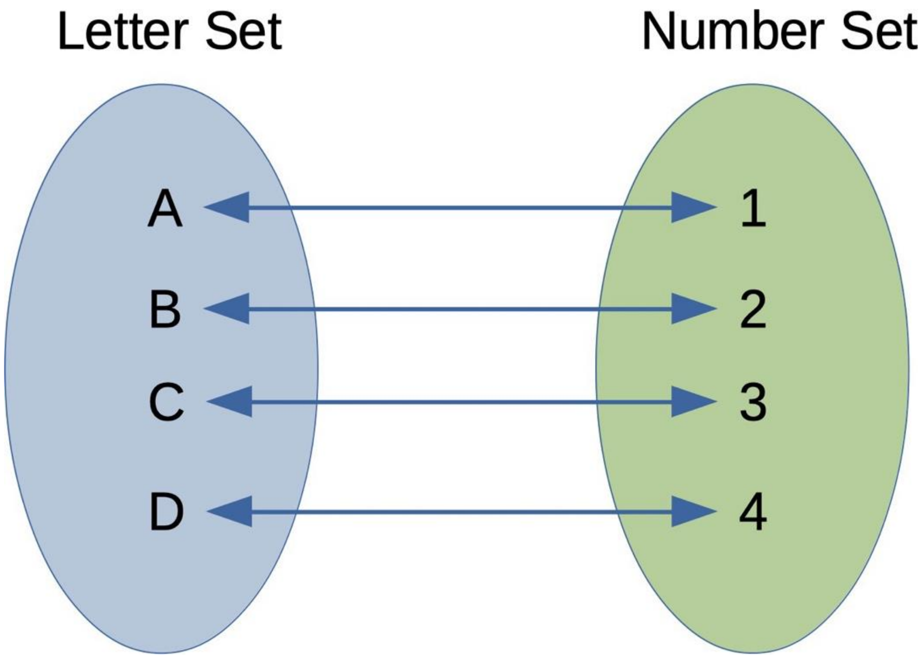
- To remember and understand some concepts about the set, exploring how its use can simplify the implementation of programs.

Georg Ferdinand Ludwig Philipp Cantor

- German mathematician: 19 February 1845 - 6 January 1918
- Function definition: established the importance of one-to-one correspondence between the members of two sets (more on that in a moment!)
- Defined infinite and well-ordered sets
- Proved that the real numbers (*rational* and *irrational*) are more numerous than the natural numbers (*counting* numbers)

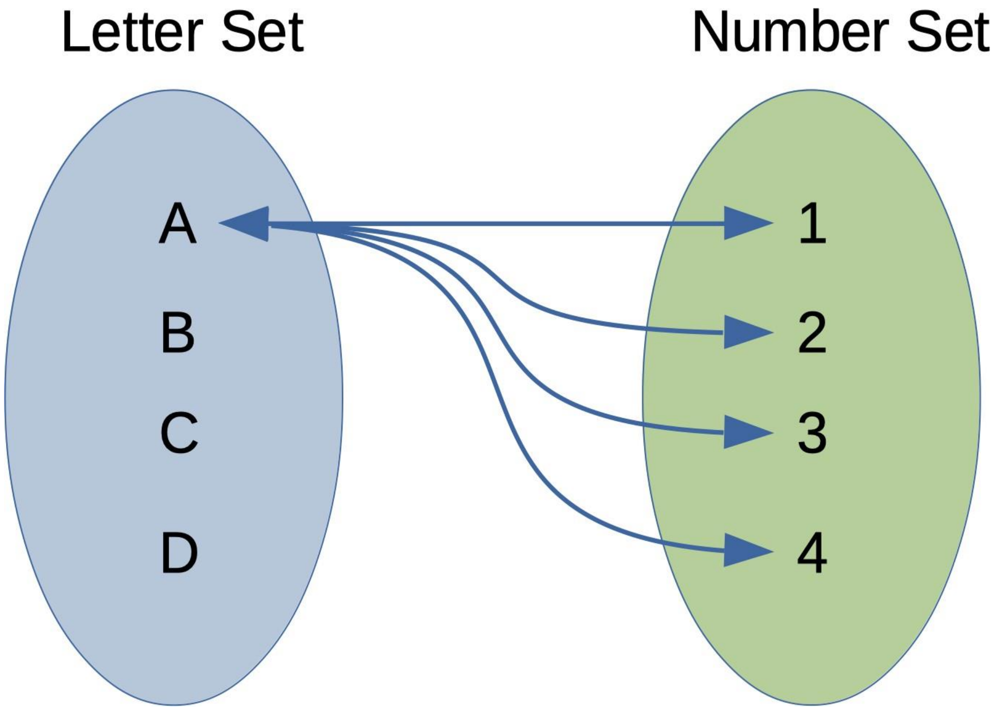
Functions as Sets

Regular Set: one-to-one relationship maintained



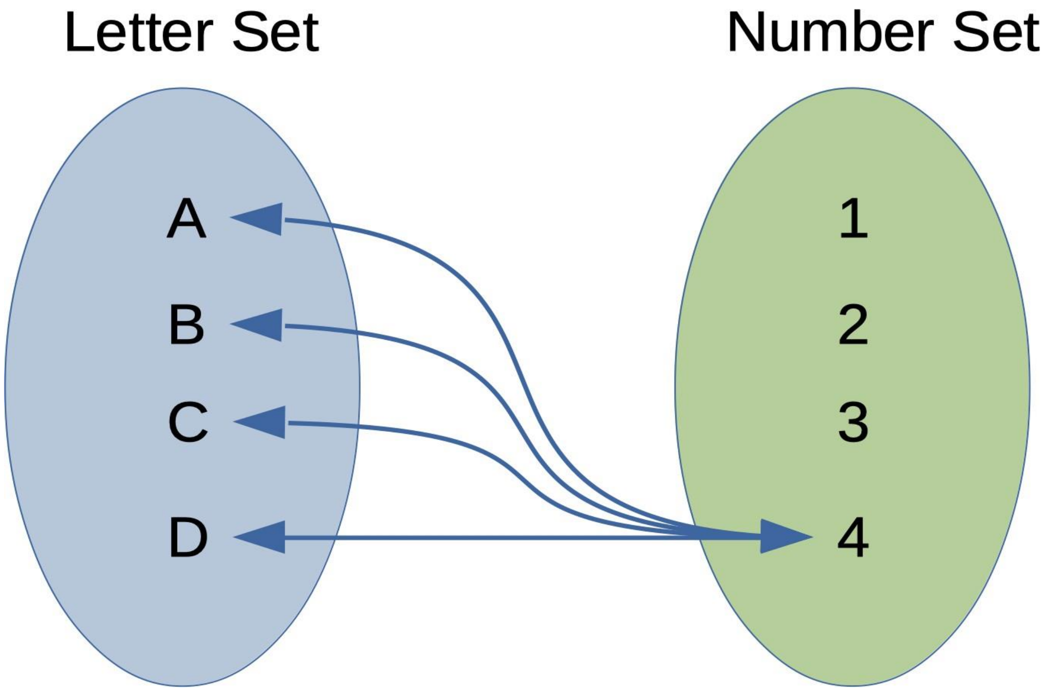
- The Letter set maps to the Number set.
- $LetterSet(x) \rightarrow NumberSet(y)$

Functions Sets



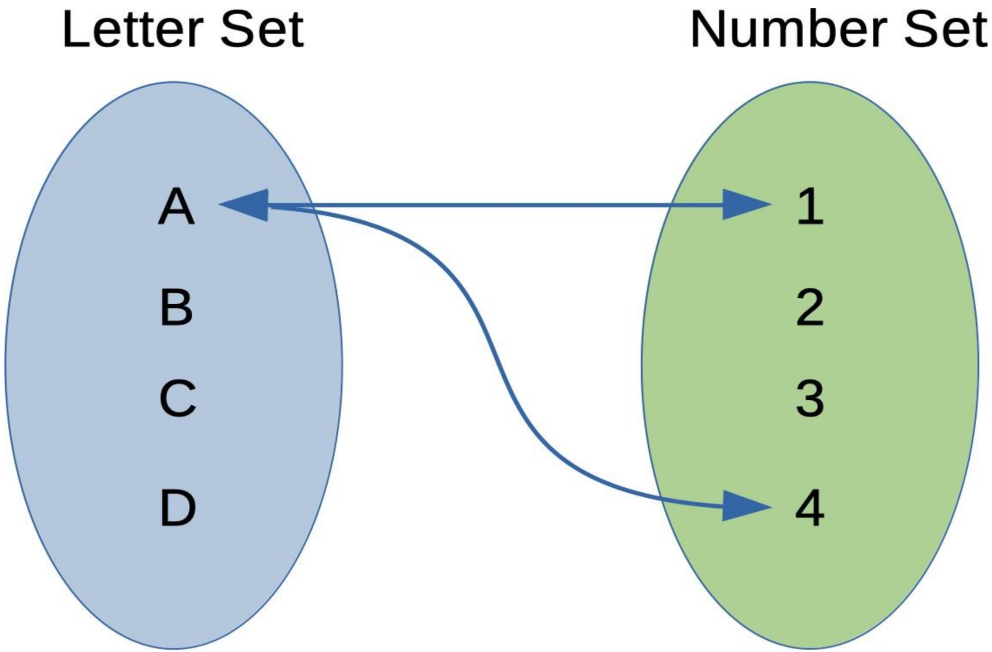
- The Letter set maps to the Number set.
- $LetterSet(x) \rightarrow NumberSet$

Functions as Sets



- Multiple elements of Letter set map to Number set.

Functions as Sets



- Multiple elements of Number set map to Letter set.

General Sets

What is a set?

- For example, the numbers 1, 2, and 3 are distinct objects when considered separately, but when they are considered **collectively,** they form a single set of size three, written {1,2,3}.
- Set theory is now a ubiquitous part of mathematics,
- May be used as a foundation from which nearly all of mathematics can be derived (From 19th century mathematical thinking!)

Types of Sets

Intentional and Extensional

Question: What kind of set do we have?

Answer: We can provide two main definitions of sets.

Intentional definition of sets: *I intend this set to be …*

- Defines a set by specifying the necessary and sufficient conditions for when the set should be used.

Extensional definition of sets: *Logically this set is …*

- Defines a set by some definition of a concept or a term.

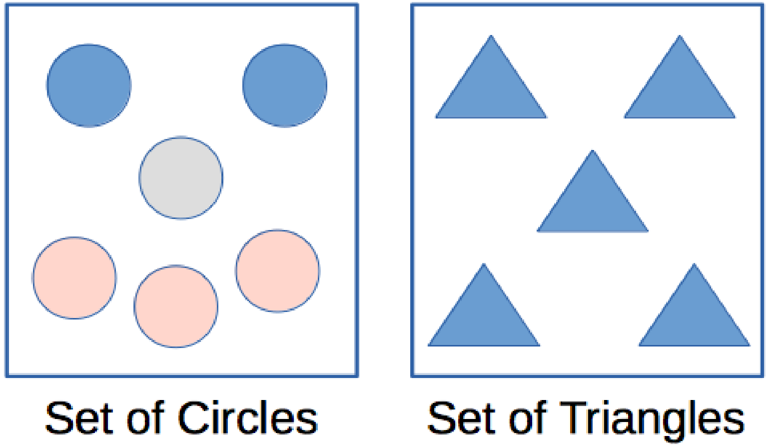
Types of Sets

A list of characters in Sherlock Holmes

- {Sherlock Holmes, Dr. John Watson, D.I. Greg Lestrade, Mrs. Hudson, Mycroft Holmes, Irene Adler, Mary (Morstan) Watson}

Types of Sets

Intentional: One decides which elements make up a set



Intentional definition of sets: *I intend that these set be …*

- The set of blue, grey and pink circles
- The set of blue triangles
- The set of colors of the Union Jack (i.e., the British flag)

Types of Sets

Extensional: Sets of members in curly brackets

Extensional definition of sets

- $A_2 = \{4, 2, 1, 3\}$
  - The first four positive numbers
- $B_2 = \{\text{Blue, Red and White}\}$
  - The set of colors of the Union Jack (the British flag)

Types of Sets

Extensional definition of sets: a list of its members in curly brackets

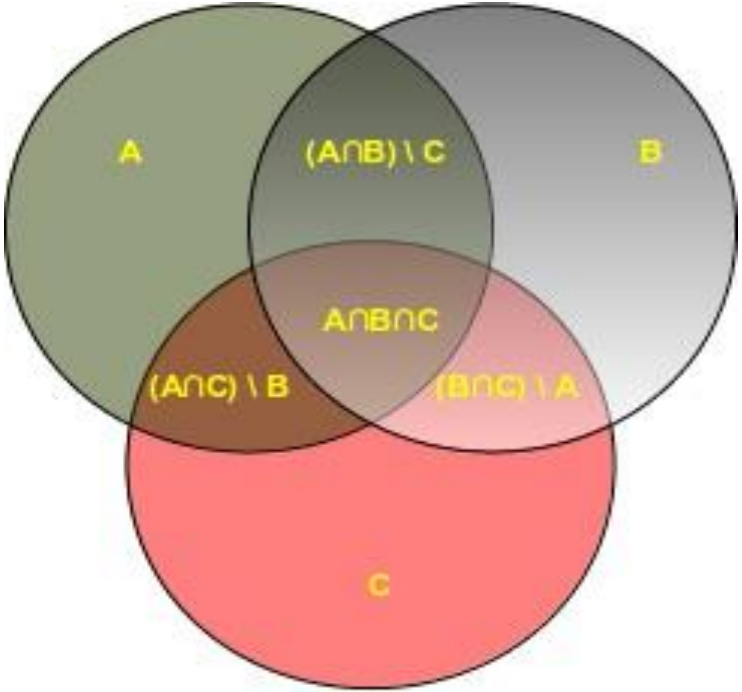
- **Intentional Definition:**
  - A_1 is the set are the first four positive integers.
  - B_1 is the set of colors of the Union Jack
- **Extensional Definition:**
  - $A_2 = \{4, 2, 1, 3\}$
  - $B_2 = \{$Blue, Red and White$\}$

Specify a set *intentionally* or *extensionally*

- In the examples above, for instance, $A_1 = A_2$ and $B_1 = B_2$

Sets with Notation

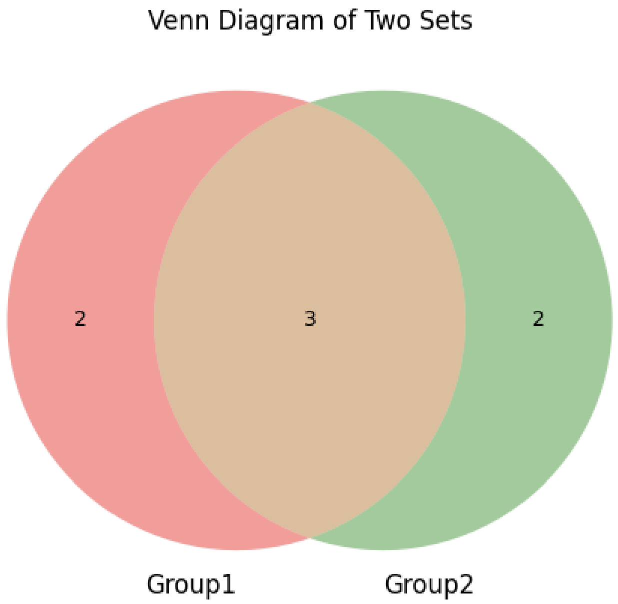
Venn Diagram



- ∪, Union: $A \cup B$ of a collection of sets A and B is the set of all elements in the collection
- ∩, Intersection $A \cap B$ of two sets A and B is the set that contains all elements of A that also belong to B

Create your own Venn diagram of TWO sets!!

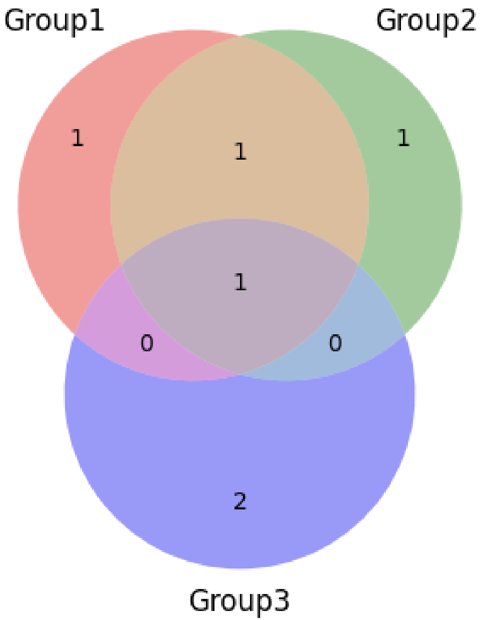
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# ---------------------------------------------------
# setup a python virtual environment
# python3 -m venv myVenv
# source myVenv/bin/activate # macOS
# myenv\Scripts\activate     # Windows
# pip install matplotlib_venn

import matplotlib.pyplot as plt
from matplotlib_venn import venn2

# Define the two sets
set1 = set([1, 2, 3, 4, 5])
set2 = set([3, 4, 5, 6, 7])
# Create a Venn diagram
venn2([set1, set2],('Group1', 'Group2'))
# Add a title
plt.title('Venn Diagram of Two Sets')
# Show the plot
plt.show()
```



Note: you may need to run this code in a virtual environment with numpy and matplotlib installed!

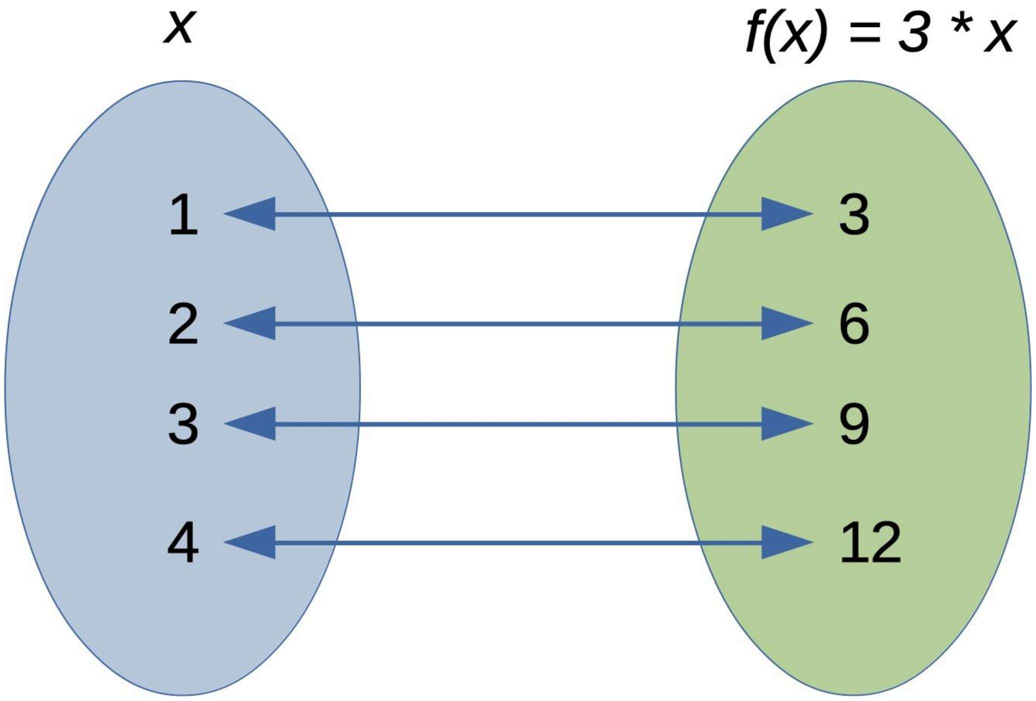
Create your own Venn diagram of THREE sets!!

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# ----------------------------------------------------
# setup a python virtual environment
# python3 -m venv myVenv
# source myVenv/bin/activate
# pip install matplotlib_venn

import matplotlib.pyplot as plt
from matplotlib_venn import venn3

set1 = set(['A', 'B', 'C'])
set2 = set(['A', 'B', 'D'])
set3 = set(['A', 'E', 'F'])

venn3([set1, set2, set3], ('Group1', 'Group2', 'Group3'))

plt.show()
```



Note: you may need to run this code in a virtual environment with numpy and matplotlib installed!

Function-based Set Transformation



Infinite Sets

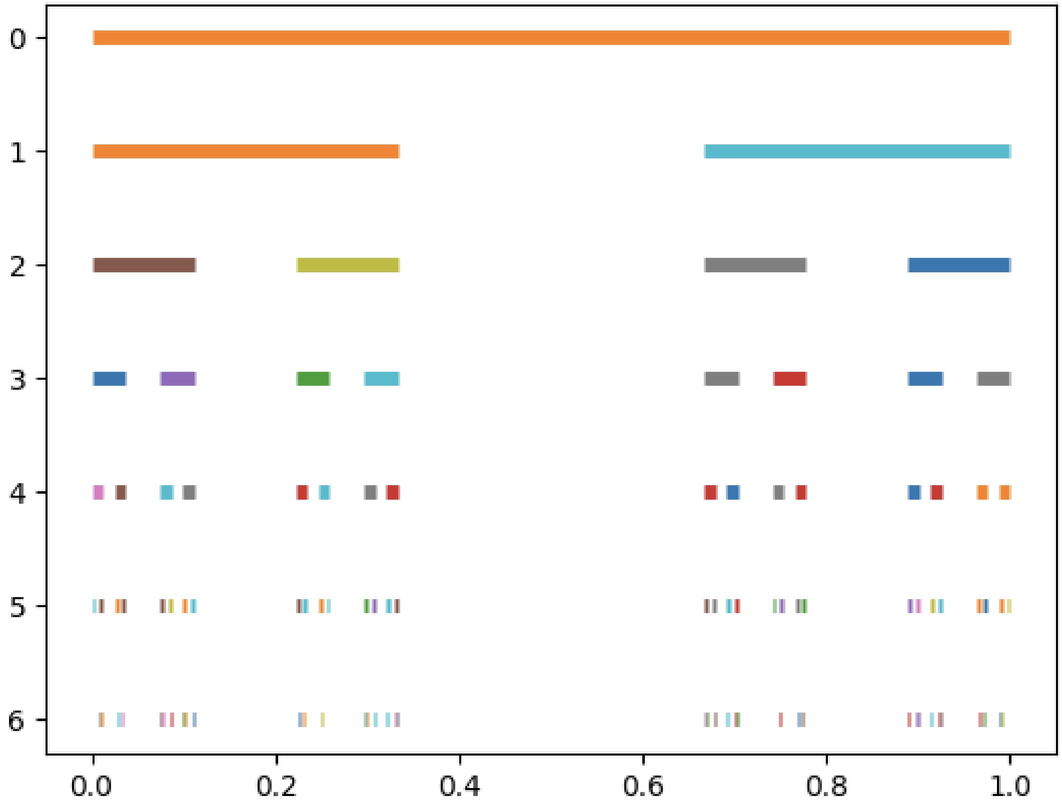
See File sandbox/cantorSet.py

Create your own Cantor set!!

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import random
# --------------------------------------------------
myColors = mcolors.TABLEAU_COLORS
line = [0,1]
depth = 6

def divide(line, level=0):
        """ partition the lines to form the sets. """
        # thisColour = "k" # black
        thisColour = random.choice(list(myColors.values()))
        plt.plot(line,[level,level], color=thisColour, lw=5, solid_capstyle="butt")
        if level < depth:
                s = np.linspace(line[0],line[1],4)
                divide(s[:2], level+1)
                divide(s[2:], level+1)

divide(line)
plt.gca().invert_yaxis()
plt.show()
```

Note: you may need to run this code in a virtual environment with numpy and matplotlib installed!

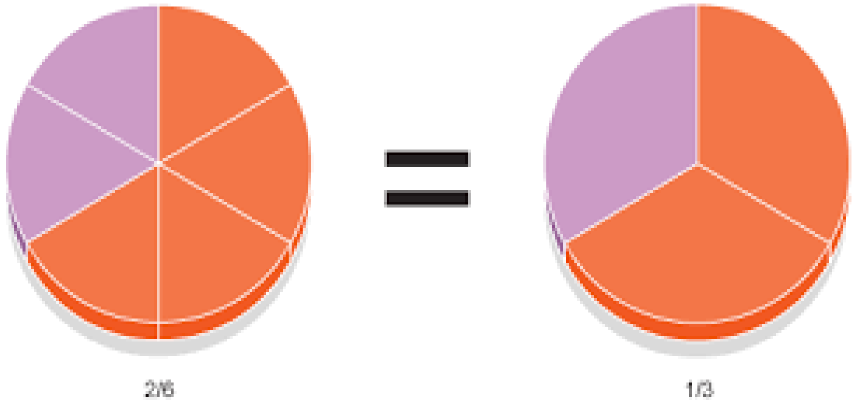
Create your own Cantor set!!



Listing Elements in Sets

- In extensionally defined sets, members in braces can be listed two or more times,
  - For example, {11, 6, 6} is identical to the set {11, 6}
- Order of members is not important
  - For example, {6, 11} = {11, 6} = {11, 6, 6, 11}

Like the equivalence of these pie charts: the content is the same in both cases



Sets in Python

An array of non-redundant elements

Creating a set of chars

```
x_st = set("This is a set")
x_st # or print(x_st)
                # the unordered chars are the elements
                # {'s', 'T', ' ', 'e', 't', 'h', 'i', 'a'}
print(type(x_st))
# <class 'set'>
```

Creating a set of string(s)

```
x_st = set(["This is a set"])
x_st # or print(x_st)
                # only one element in set; the string itself
                # {'This is a set'}
x_st = set(["This", "is", "a", "set"])
                # each word is an element
                # {'This', 'is', 'set', 'a'}
```

Sets in Python

```
# next line on one line
cities_st = set(("Paris", "Lyon", "London", "Berlin", "Birmingham", "Paris"))
print(cities_st)
                # {'Berlin', 'Paris', 'Birmingham', 'London', 'Lyon'}
```

Adding new elements

```
cities_st = set(["Frankfurt", "Basel", "Freiburg"])
cities_st.add("Meadville")
print(cities_st)
                # {'Freiburg', 'Meadville', 'Basel', 'Frankfurt'}
```

Sets in Python

Removing elements

```
cities_st = set(["Frankfurt", "Basel", "Meadville"])
cities_st.remove("Meadville") # Meadville is a key
print(cities_st)
          # {'Basel', 'Frankfurt'}
```

Frozensets cannot be changed

```
cities_st = frozenset(["Frankfurt", "Basel", "Freiburg"])
cities_st.add("Meadville")
          # AttributeError:
          # 'frozenset' object has no attribute 'add'
print(cities_st)
          # frozenset({'Freiburg', 'Basel', 'Frankfurt'})
type(cities_st)
          # <class 'frozenset'>
```

Sets in Python

Removing all elements of set

```
cities_st = {"Stuttgart", "Konstanz", "Freiburg"}
print(cities_st)
                # {'Freiburg', 'Konstanz', 'Stuttgart'}
cities_st.clear()
print(cities_st)
                # set()
```

Determining difference between sets

```
x = {"a","b","c","d","e"}
y = {"b","c"}
z = {"c","d"}
print(x.difference(y)) # {'a', 'e', 'd'}
print(x.difference(y).difference(z)) # {'a', 'e'}
```

- Returns the characters which are never repeated across {x, y, z}

Sets in Python

Difference and subtraction

```
x = {'c', 'a', 'd', 'b', 'e'}
y = {'c', 'b'}
x.difference_update(y)
print(x) # {'a', 'd', 'e'}
print(y) # {'c', 'b’}
```

```
x = {"a","b","c","d","e"}
y = {"b","c"}
x = x - y
print(x) # {'e', 'd', 'a'}
```

- Top: Returns an updated set of x of the characters which are never repeated across {x, y}

Sets in Python

Cloning and removing from original

```
x = {'e', 'd', 'a'}
v = x
print(x) # {'a', 'e', 'd'}
print(v) # {'a', 'e', 'd'}
x.remove('a')
x # {'e', 'd'}
v # {'e', 'd'}

v.remove('d')
print(x) # {'e'}
print(v) # {'e'}
```

- x = v does not make a copy of x . Instead, this is a reference from one object to another.

Checking for Particular Elements

Is an element in a List?

```
x = {"a","b","c","d","e"}
print("e" in x) # True
print("e" and "a" in x) # True
print("e" and "i" in x) # False
print("i" and "e" in x)
print("i" in x and "e" in x)
```

Iterating Through Elements in Sets

Iteration

```
abc_set = {"a","b","c","d","e"}
for i in abc_set:
        print(i)
```

Note: Since there is no order control in the set, you cannot know which element will be printed first (from above).

Creating Solutions

Go check out the fun code about sets in the sandbox/!