Discrete Structures: CMPSC 102

Oliver BONHAM-CARTER

Fall 2022

Week 12

Let's Discuss

Key Questions

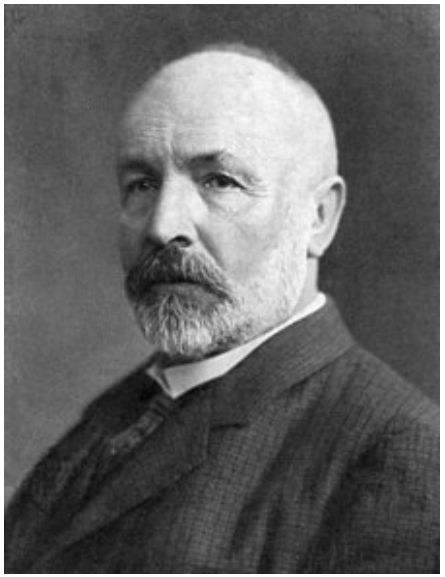
How do I use the mathematical concepts of **sets** and **Boolean logic** to design Python programs that are easier to implement and understand?

Learning Objectives

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

Georg Ferdinand Ludwig Philipp Cantor

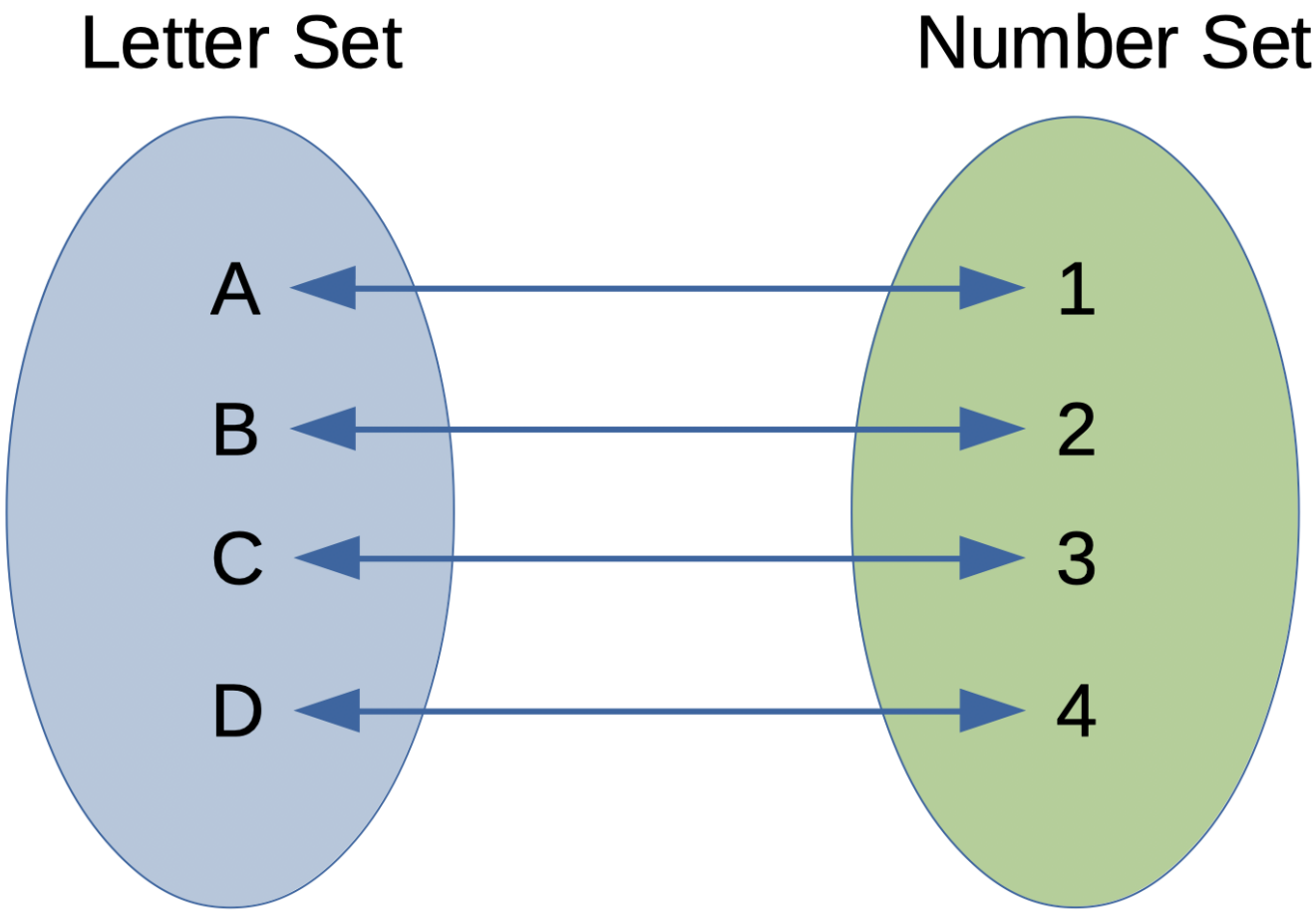
Creator of Set theory



- 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

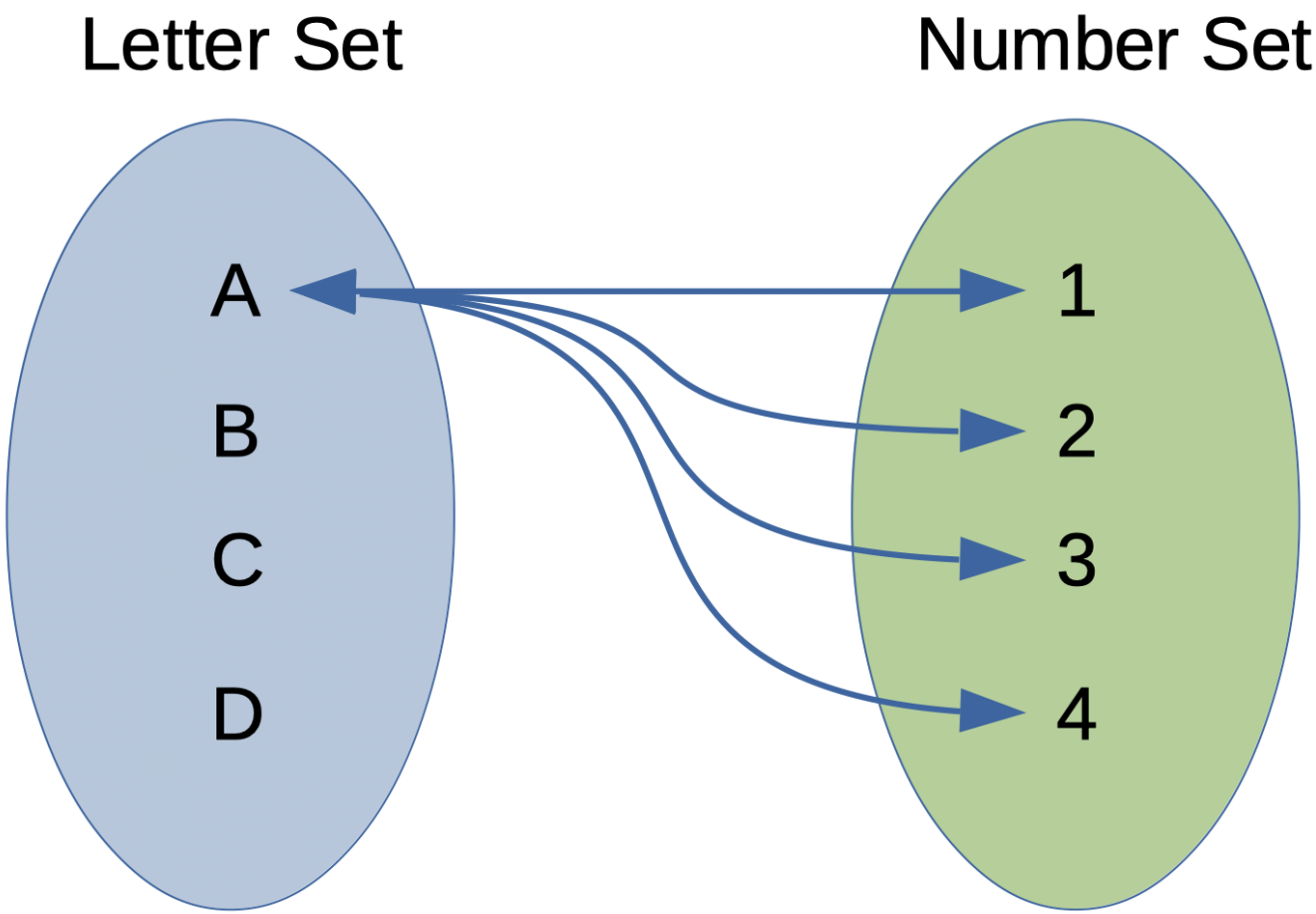
Regular Set: one-to-one relationship maintained



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

Functions Sets

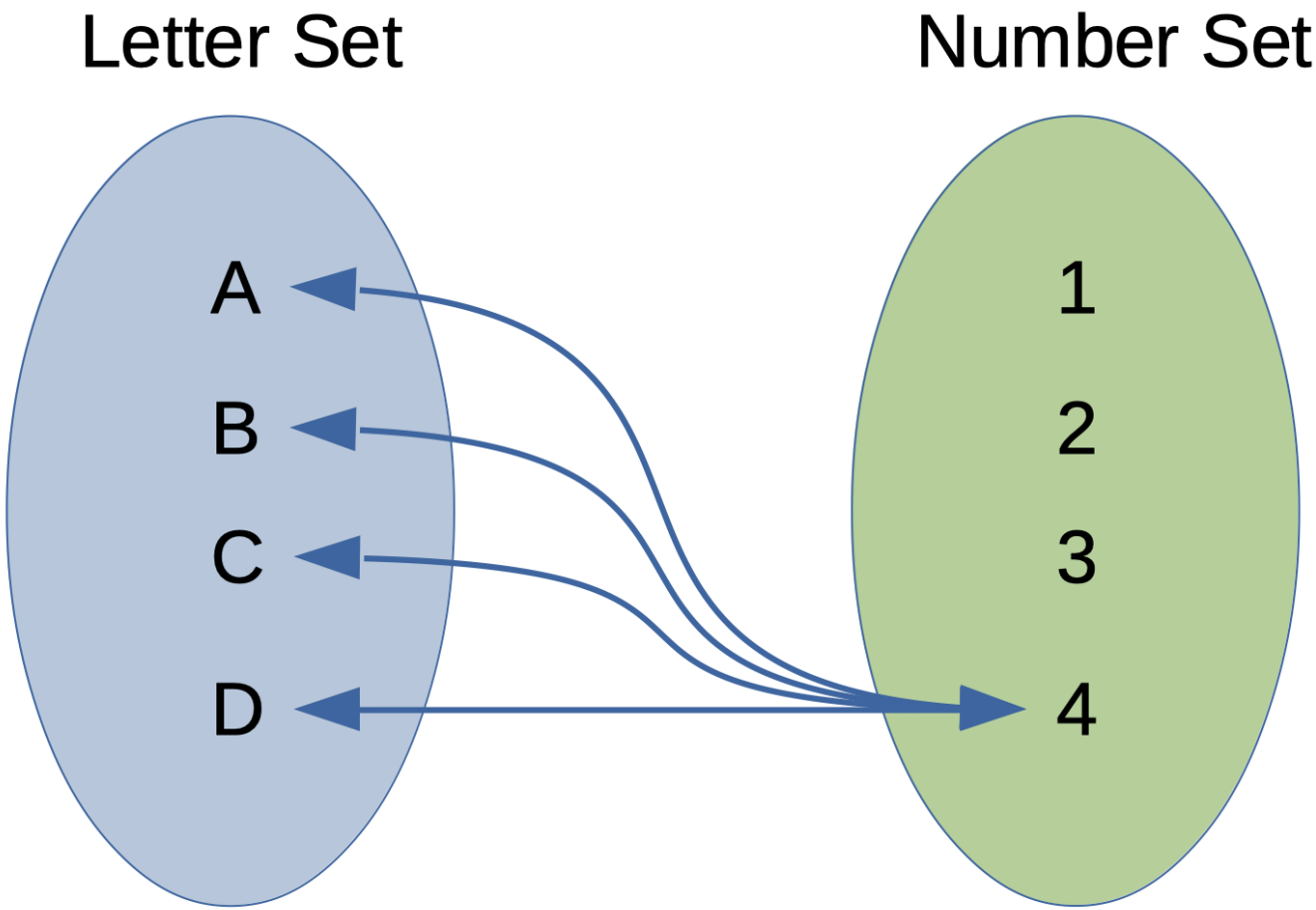
Regular Set: one-to-one-*ism* is maintained



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

Functions as Sets

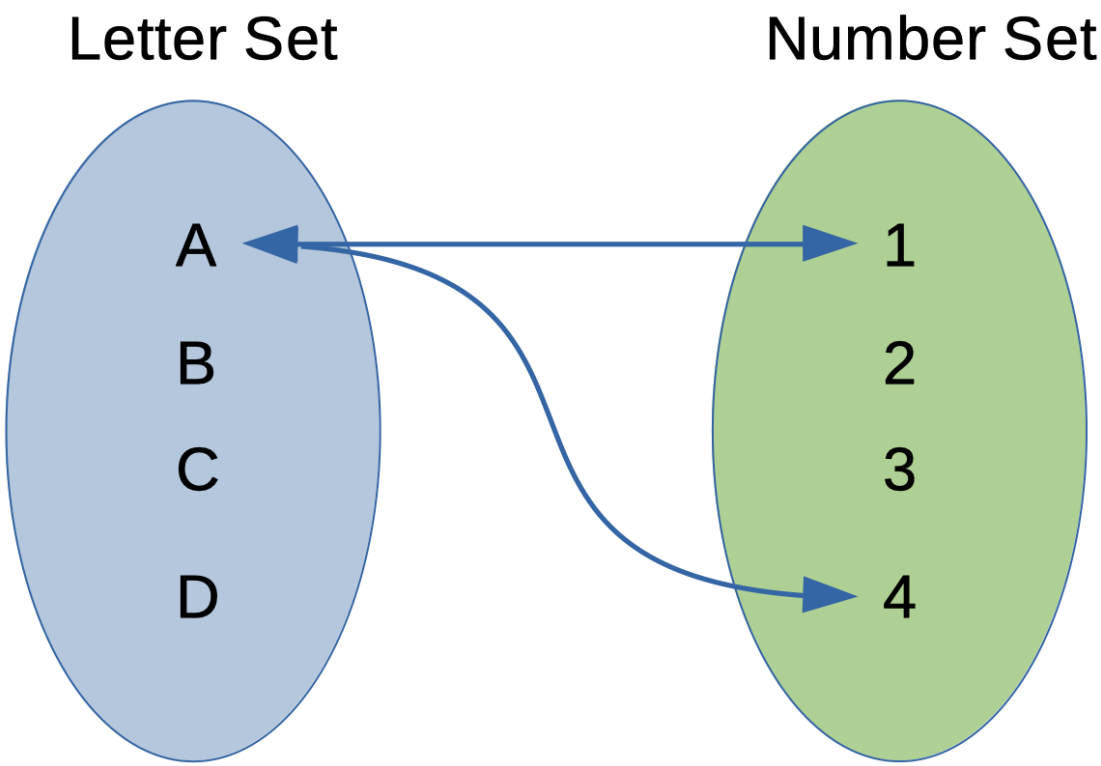
One-to-one-*ism* is NOT maintained!



- Multiple elements of Number set map to Letter set.

Functions as Sets

One-to-one-*ism* is NOT maintained!



- 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 19^{th} century mathematical thinking!)

Types of Sets
Intensional 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.

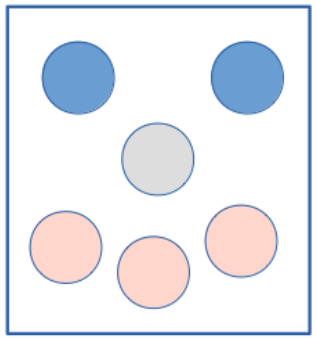
Intensional Sets

A list of characters in Sherlock Holmes

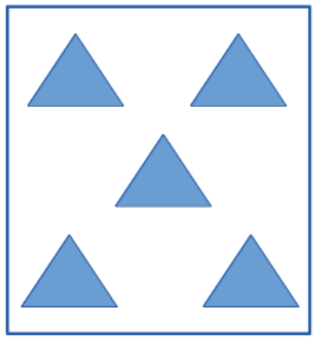
- {Sherlock Holmes, Dr. John Watson, D.I. Greg Lestrade, Mrs. Hudson, Mycroft Holmes, Irene Adler, Mary (Morstan) Watson}

Types of Sets

Intentional: One decides which elements make up a set



Set of Circles



Set of Triangles

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)
- $F = \{n^2 - 4 : n \text{ is an integer; and } 0 \leq n \leq 19\}$
  - The set of all values gained from plugging in n between 0 and 19 into the equation $n^2 - 4$

Types of Sets

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

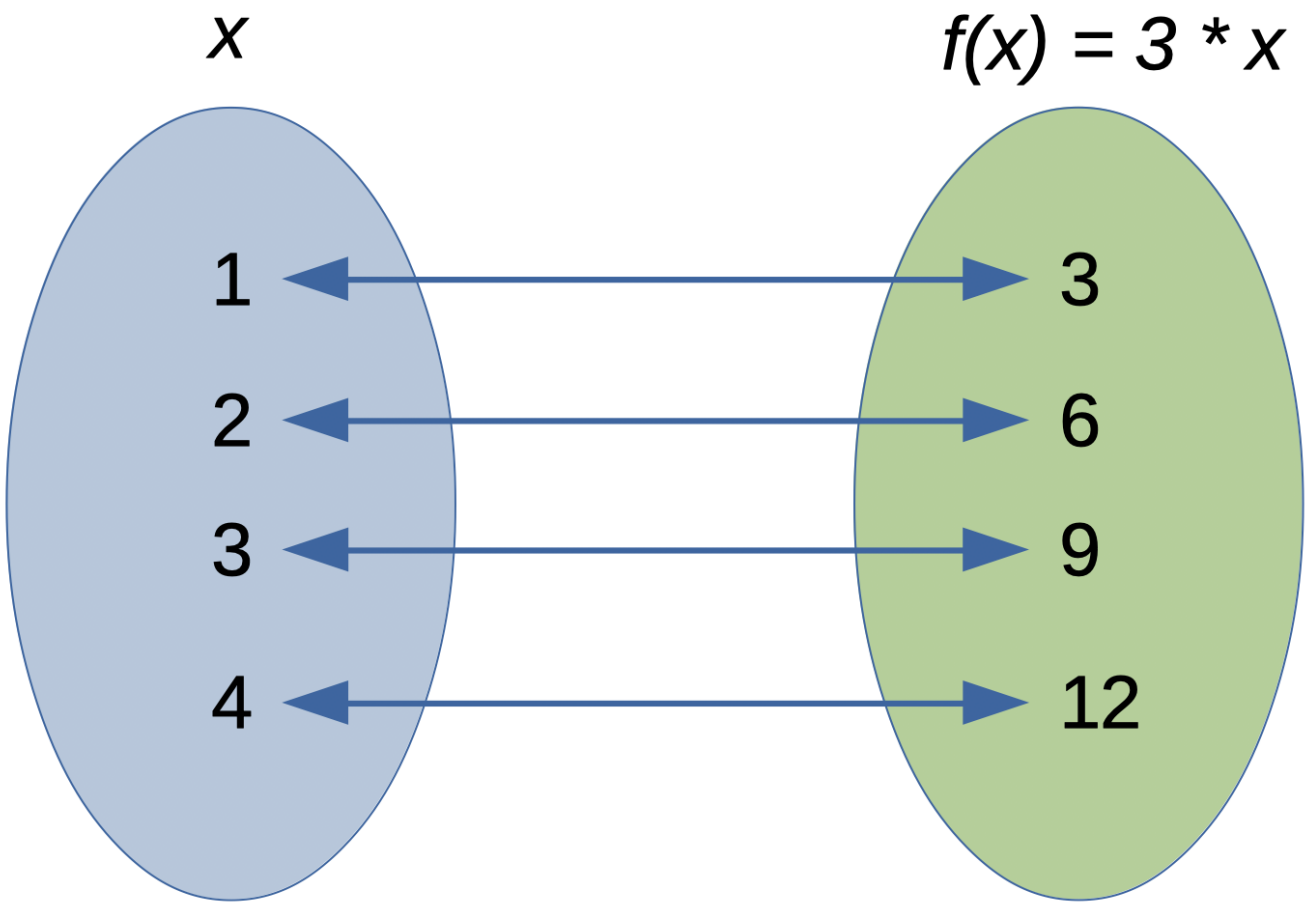
- **Intentional Definition:**
  - A_1 is the set are the first four positive integers.
  - B_1 is the set of colors of the Union Jack
- **Extensional Definition:**
  - $A_2 = \{4, 2, 1, 3\}$
  - $B_2 = \{\text{Blue, Red and White}\}$

Specify a set *intensionally* or *extensionally*

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

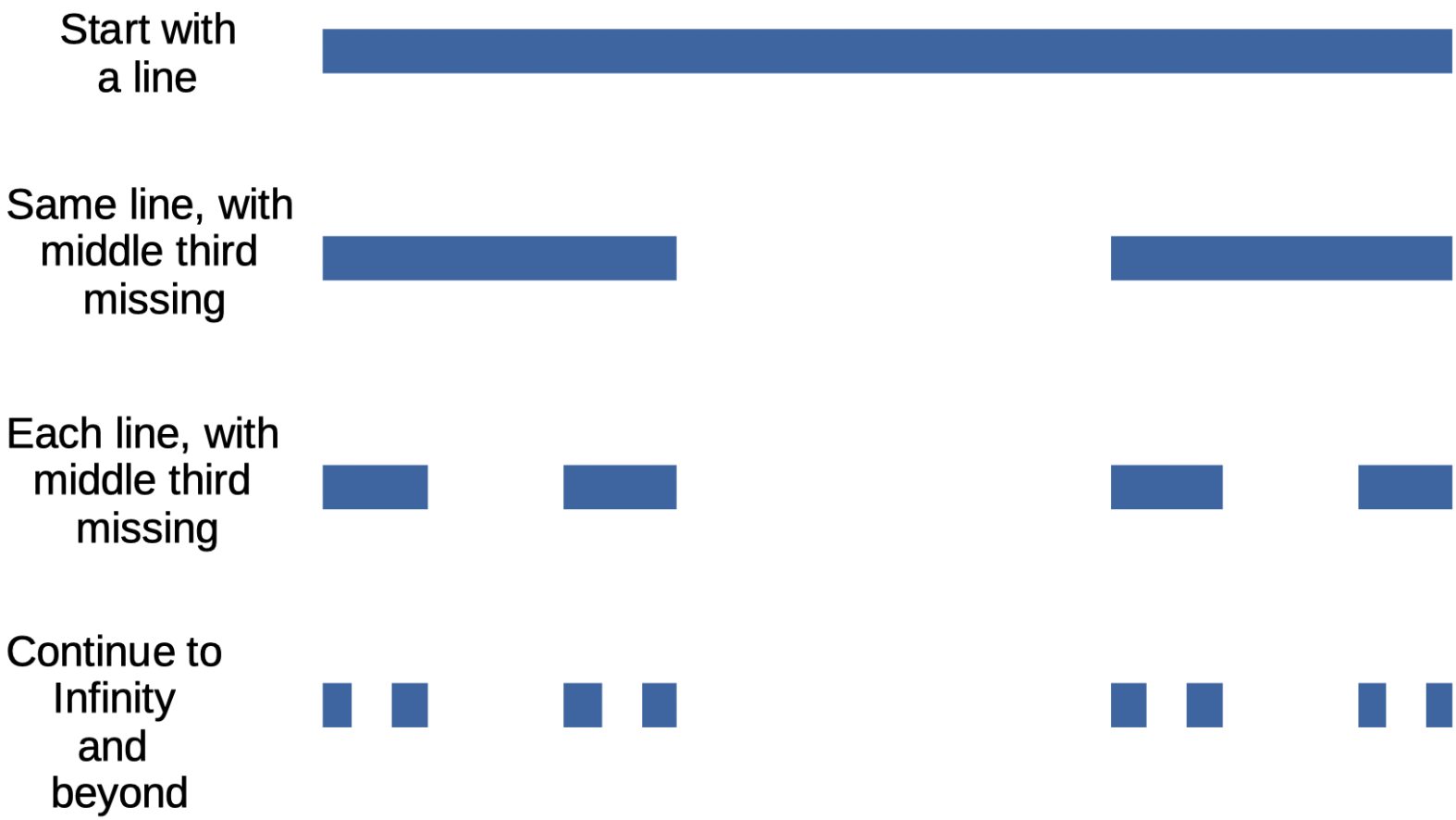
Infinite Sets: an Extensional set example

Sets that go on forever



Infinite Sets: an Extensional set example

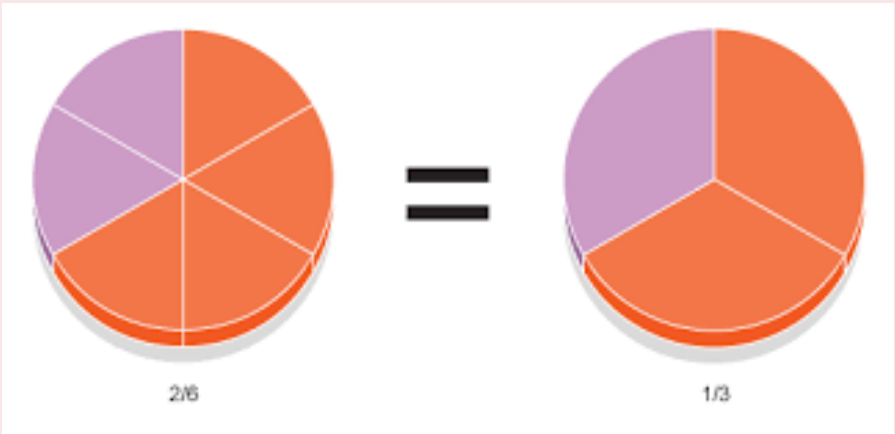
See File `sandbox/cantorSet.py`



Listing Elements in Sets

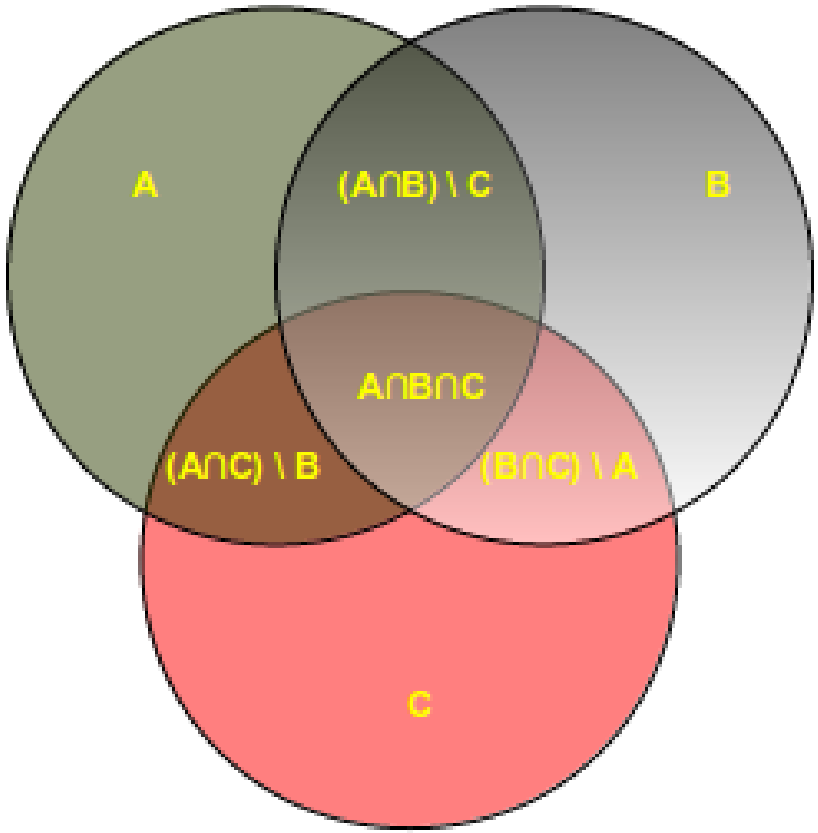
- In extensionally defined sets, members in braces can be listed two or more times,
  - For example, $\{11, 6, 6\}$ is identical to the set $\{11, 6\}$
- Order of members is not important
  - For example, $\{6, 11\} = \{11, 6\} = \{11, 6, 6, 11\}$

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



Sets with Notation

Venn Diagram



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

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

Determining difference between sets

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

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

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'}

print(x)  # {'a', 'e', 'd'}
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 $\{x, y, y\}$

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')
x   # {'e'}
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"}
"e" in x    # True
"e" and "a" in x  # True
"e" and "i" in x  # False
```

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).