

Introduction to Sets

[_ \(https://github.com/learn-co-curriculum/dsc-intro-to-sets\)](https://github.com/learn-co-curriculum/dsc-intro-to-sets)[_ \(https://github.com/learn-co-curriculum/dsc-intro-to-sets/issues/new/choose\)](https://github.com/learn-co-curriculum/dsc-intro-to-sets/issues/new/choose)

Introduction

You have definitely heard of sets before. In this section, however, you will learn about the formal definition of sets, which will serve as a foundation for everything related to probability and combinatorics!

Objectives

You will be able to:

- Define a set in the context of probability theory
- Define a universal set and subsets
- Describe the process of making unions, intersections, and complements
- Use Venn Diagrams to visually demonstrate set operations
- Describe the inclusion-exclusion principle

What is a Set?

In probability theory, a set is defined as a *well-defined collection of distinct objects*. In other words, it contains 0 or more unique items, and is defined unambiguously, so that it is clear whether any given item is an element of the set.

Set Definitions and Notations

Mathematically, you can denote a set by S . Typically the names of sets are written as italicized capital letters (in markdown, this set is written by surrounding it with dollar signs, e.g. S for the example from the previous sentence).

There are a couple primary ways we will define sets:

Semantic Definition

When discussing sets from a theoretical perspective, we will often define a set with a written definition, rather than using math notation. For example:

A is defined as the set of even numbers



Help

Enumeration

Another way to define a set is by listing its elements, surrounded by curly braces. For example:

$$S = 1, 2, 3$$

Note that order does not matter in defining a set. " $S = 1, 2, 3$ " and " $S = 3, 2, 1$ " mean the same thing.

When working with sets in Python, we will frequently use this form of defining sets, since Python is not designed to work with infinitely-large collections. The below code snippet shows S defined in Python:

```
S = {1, 2, 3}
type(S)
```

```
set
```

Set Membership

Unlike other types of collections (e.g. a sequence in math, or a list in Python), sets are chiefly concerned with *membership*, i.e. whether a given element belongs to the collection. Order doesn't matter, and elements can only appear once in a set.

Membership Notation

If an element x belongs to a set S , then you'd write $x \in S$. On the other hand, if x does not belong to a set S , then you'd write $x \notin S$.

Example: If S is defined as the set of even numbers, then:

- If $x = 2$, $x \in S$ because x is an even number.
- If $x = 9$, $x \notin S$ because x is not an even number.

In Python, we can use the familiar `in` operator, optionally negating it with `not`. Returning to the previously-defined set `S`:

```
1 in S
```

```
True
```

```
}
```

 Help

False

Other Important Set Definitions

Universal Sets

The collection of all possible outcomes in a certain context or universe is called the **universal set**. A universal set is often denoted by Ω .

Example of a universal set: All the possible outcomes when rolling a dice.

$$\Omega = 1, 2, 3, 4, 5, 6$$

Remember that a universal set is not necessarily all the possible things that have ever existed. Typically, a universal set is just all the possible elements within certain bounds, e.g., the set of all countries in the world, the set of all the animal species in the Bronx Zoo, etc.

A universal set can have an infinite number of elements — for example, the set of all real numbers!

Empty Sets

When there are no elements in a certain set, this set is **empty**, denoted by \emptyset or simply

Working with Multiple Sets

Subsets

Set T is a subset of set S if *every element* in set T is also in set S . The mathematical notation for a subset is $T \subseteq S$.

For example, say we have sets A , B , and C :

$$A = 1, 2, 3$$

$$B = 1, 2, 3$$

$$C = 1, 2$$

Intuitively, you probably understand that $C \subseteq A$ and $C \subseteq B$ (i.e. C is a subset of A and C is a subset of B).

What might be less intuitive is that $A \subseteq B$ and $B \subseteq A$ (i.e. A is a subset of B and B is a subset of A). According to our formal definition, if two sets contain the same elements, they are each a subset of the other.

Learn more examples in Python:



```
# Setting up example sets
```

```
A = {1, 2, 3}
```

```
B = {1, 2, 3}
```

```
C = {1, 2}
```

```
# Intuitively makes sense, C is a subset of A
```

```
C.issubset(A)
```

```
True
```

```
# Somewhat less intuitive, B is also a subset of A
```

```
# even though B and A contain the same elements
```

```
B.issubset(A)
```

```
True
```

We can also use `<=` to check if something is a subset:

```
# We're asking "is C a subset of A?"
```

```
C <= A
```

```
True
```

```
# We're asking "is B a subset of A?"
```

```
B <= A
```

```
True
```

Proper Subsets

If you don't want to include sets that contain the exact same elements, you are looking for a *proper subset*.

Set T is a proper subset of set S if every element in set T is also in set S *and the two sets do not contain the exact same elements*. The mathematical notation for a proper subset is $T \subset S$.

For example, if S is the set of even numbers, set $T = \{2, 6, 22\}$ is a proper subset of S . All values in T are even numbers, but there are many additional even numbers that are not in T .

All proper subsets are subsets, but not all subsets are proper subsets. Returning to A , B , and C from [Help](#), therefore also $C \subseteq A$ (i.e. C is a proper subset of A , therefore C is also a

subset of A). However, even though it is true that $B \subseteq A$, it is not true that $B \subset A$ (i.e. B is a subset of A , but B is not a proper subset of A , because A and B contain the exact same elements).

In Python, there is no method for finding a proper subset, but we can just use the `<` operator.

```
# We're asking "is C a proper subset of A?"
C < A
# True, because every element of C is in A,
# and A contains at least 1 element that C does not
```

True

```
# We're asking "is A a proper subset of B?"
A < B
# False, because even though every element of A is in B,
# A and B have the exact same elements
```

False

The Python notation helps to make this distinction clearer. A subset (`<=`) includes sets that are "equal" (have the exact same elements), whereas a proper subset (`<`) only includes sets that have fewer elements. The `<` and `<=` operators are being used differently than you have seen previously (e.g. `4 < 5`) but there is still a conceptual relationship between them.

Supersets

Another term you might see is a *superset*. A superset is just the inverse of a subset. For example, A is a superset of C . Or in math notation, $A \supseteq C$.

Just like with a subset, a proper superset is one where the two sets are not equal. So, $A \supset C$ and $A \supseteq C$ (i.e. A is a superset as well as a proper superset of C), but it's only true that $A \supseteq B$, not that $A \supset B$ (i.e. A is a superset of B but not a proper superset).

In Python this can be represented using the `>` and `>=` operators as well as the `.issuperset` method. `.issuperset` corresponds with `>=` so it's useful to be familiar with both forms of the notation.

```
# We're asking "is A a superset of C?"
A >= C
```

True



```
# We're asking "is A a proper superset of C?"
```

```
A > C
```

```
True
```

```
# We're asking "is A a superset of B?"
```

```
A >= B
```

```
True
```

```
# We're asking "is A a proper superset of B?"
```

```
A > B
```

```
False
```

```
# You can also use this method if you're only checking
```

```
# for a superset, not a proper superset
```

```
A.issuperset(C)
```

```
True
```

Core Set Operations

Next, let's talk about set operations. Imagine you have two sets of numbers, say the first 4 multiples of 3 in set S :

$$S = 3, 6, 9, 12$$

and the first 4 multiples of 2 in set T :

$$T = 2, 4, 6, 8.$$

Below, we define these sets in Python.

```
S = {3, 6, 9, 12}
```

```
T = {2, 4, 6, 8}
```

We will also define the universal set (Ω) for these exercises as the multiples of both 2 and 3 until 20. In other words:

$$\Omega = 8, 9, 10, 12, 14, 15, 16, 18, 20$$

 Help

```
omega = {2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20}
```

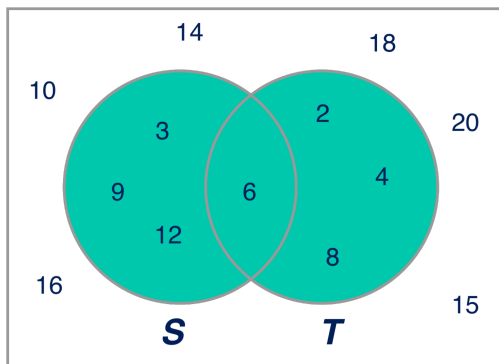
a) Union of Two Sets

The union of two sets S and T is the set of elements of either S or T , or in both.

Applied to our example, the union of S and T is given by the elements **2, 3, 4, 6, 8, 9, 12**. (Note that even is a value appeared in both of the two original sets, it only appears once in the resulting union.)

A popular way to represent sets and their relationships is through Venn Diagrams,

(https://en.wikipedia.org/wiki/Venn_diagram https://en.wikipedia.org/wiki/Venn_diagram), see picture below!



(Note that elements of Ω that are not part of S or T are outside of both circles but still within the bounding box.)

In mathematical terms, the union of S and T is denoted as $S \cup T$.

In Python, the union of two sets is calculated using the `|` operator:

```
S | T
```

```
{2, 3, 4, 6, 8, 9, 12}
```

Alternatively, the `.union` method can be used:


```
S.union(T)
```

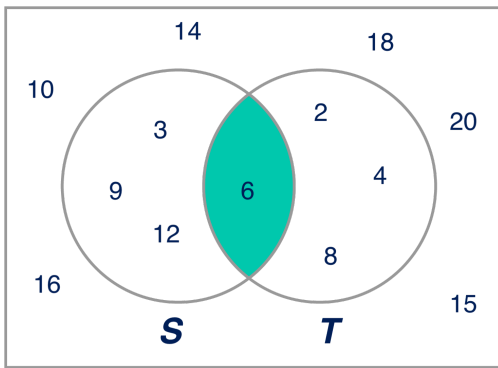
```
{2, 3, 4, 6, 8, 9, 12}
```

b) Intersection of Two Sets

The intersection of two sets S and T is the set that contains all elements of S that also belong to T .

Applied to our example, the intersection of S and T is given by **6**, so it contains the elements that

a  **Help** both 2 AND 3.



In mathematical terms, the intersection of S and T is denoted as $S \cap T$.

In Python, the intersection of two sets is calculated using the `&` operator:

```
S & T
```

```
{6}
```

Alternatively, the `.intersection` method can be used:

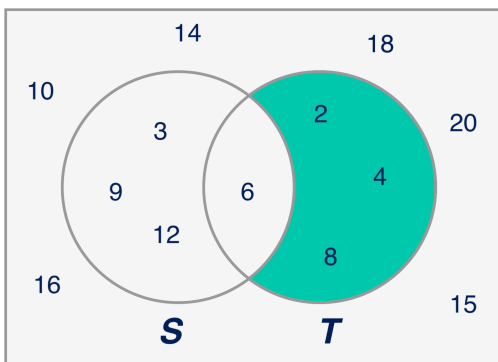
```
S.intersection(T)
```

```
{6}
```

c) Relative Complement or the Difference

In general, the *complement* of a set means the elements that are not in that set. One way to scope this is to find all of the elements of one set that are not in another set — known as the *relative complement*.

For example, the relative complement of S in T is $2, 4, 8$, meaning the set of elements that are in T but not in S . This is also referred to as the *difference* between T and S .



In mathematical terms, difference is denoted by $T \setminus S$ or $T - S$. In this case, the relative complement of S in T (or $T \setminus S$) is $2, 4, 8$. The relative complement of T in S (or $S \setminus T$) is $3, 9, 12$.

Help

In Python, the difference between two sets is calculated using the `-` operator:

$T - S$

$\{2, 4, 8\}$

$S - T$

$\{3, 9, 12\}$

Alternatively, the `.difference` method can be used:

`T.difference(S)`

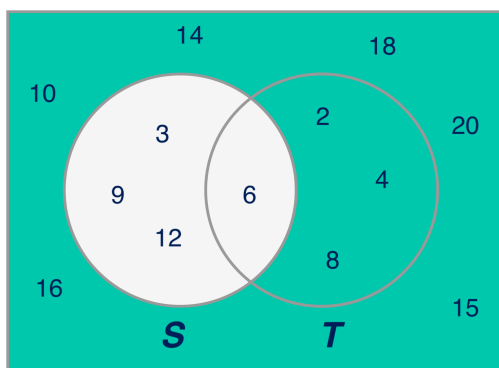
$\{2, 4, 8\}$

d) Absolute Complement

Another way to define the complement is to find all elements that are not in the universal set — known as the *absolute complement*.

The absolute complement of S , with respect to the universal set Ω , is the collection of the objects in Ω that don't belong to S .

Returning to the previous T and S example, the absolute complement of S would be $2, 4, 8, 10, 14, 15, 16, 18, 20$, as shown in the diagram below.



Mathematically, the absolute complement of S is denoted as S' or S^c . In other words, for the example above:

$$S' = 2, 4, 8, 10, 14, 15, 16, 18, 20$$

In Python, we can use the same `-` operator as we did to find the difference above, this time using `omega` :

[\(?\) Help](#)

`{2, 4, 8, 10, 14, 15, 16, 18, 20}`

Note how the definition of Ω is very important here. Imagine a set $S = \text{elephant, alligator, tiger, bear}$. The absolute complement of this set will depend on how the universal set is defined: Is Ω equal to *the animals in the Bronx Zoo*, or *the 20 most deadly animals in the world*?

Just like a set can contain infinite values (e.g. the set of all even numbers), the complement of a set can contain infinite values. This will have implications for whether or not a given set can be represented in Python.

Additional Set Attributes

Cardinality

The *cardinality* of a set is simply the number of elements in the set.

In math notation, we represent this as $|S|$.

In Python, we can use the built-in `len` function:

```
len(S)
```

```
4
```

Note: the pipe character (`|` , vertical line) is used differently in math notation vs. Python. In math notation, two vertical lines surround the name of the set to denote cardinality. In Python, a single vertical line is an operator used to find the union between two sets (as well as for other purposes such as an OR operation between two boolean masks in pandas). Often as a data scientist you will need to translate between different kinds of notation like this — make sure you are communicating with your stakeholders to ensure you understand what is meant by a given symbol!

Inclusion-Exclusion Principle

If you want to find the cardinality of the **union** of multiple sets, you can't simply add together the cardinality of each set.

Returning to the S and T example above, we know that 6 is in both sets. So if we simply add together the two numbers, 6 gets counted twice:

```
print("(Cardinality of S) + (Cardinality of T)")
print(len(S) + len(T))
```

 **Help**

(Cardinality of S) + (Cardinality of T)

8

```
print("Cardinality of (S | T)")
print(len(S | T))
```

Cardinality of $(S | T)$

7

In combinational mathematics, the inclusion-exclusion principle is a counting technique that solves this problem.

For two finite sets, the method for counting the number of elements in the union is given by:

$$|S \cup T| = |S| + |T| - |S \cap T|$$

In other words, the cardinality of the union of the two sets $(|S \cup T|)$ is:

- $|S|$ (the cardinality of S)
- plus $|T|$ (the cardinality of T)
- minus $|S \cap T|$ (the cardinality of the intersection between S and T)

To demonstrate this in Python, recall these values:

S

$\{3, 6, 9, 12\}$

T

$\{2, 4, 6, 8\}$

$S \ \& \ T$

$\{6\}$

Now let's compute the cardinality of $|S \cup T|$, following that formula.

So, the cardinality of S :

```
len(S)
```

 **Help**

4

Plus the cardinality of T :

$$\text{len}(S) + \text{len}(T)$$

8

Minus the cardinality of $|S \cap T|$:

$$\text{len}(S) + \text{len}(T) - \text{len}(S \cap T)$$

7

Double-checking this answer:

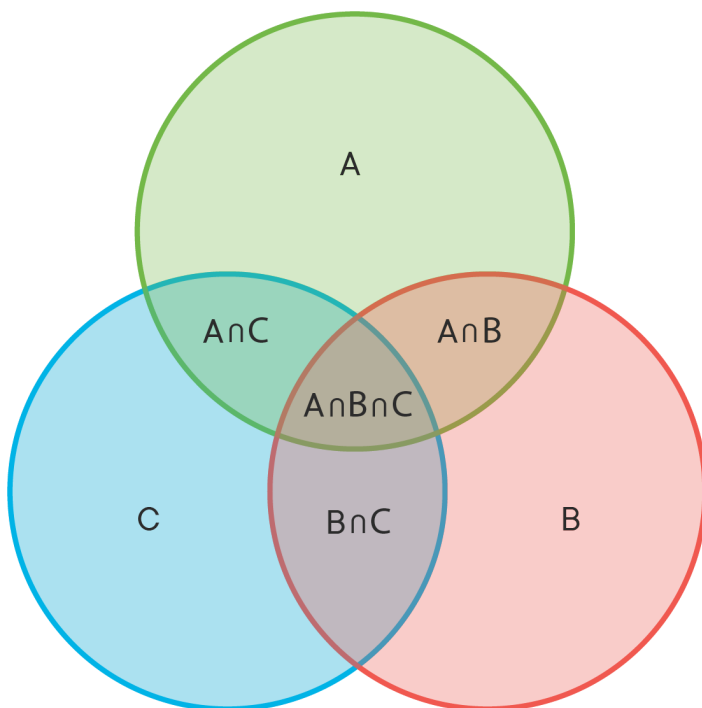
$$\text{len}(S \cup T) == \text{len}(S) + \text{len}(T) - \text{len}(S \cap T)$$

True

Great!

This formula can also be extended to three sets, four sets, etc. For example, imagine you have a third set R . The number of elements in the union of three finite sets is given by:

$$|S \cup T \cup R| = |S| + |T| + |R| - |S \cap T| - |S \cap R| - |R \cap T| + |S \cap T \cap R|$$



T is an important foundation for the probability and combinatorics concepts in the up-ns.

Sets in Python

Some things to bear in mind when working with sets in Python:

- Sets are unordered collections of unique elements.
- Sets are iterable.
- Sets are collections of lower level Python objects (just like lists or dictionaries).
- Some sets that can be represented with mathematical notation cannot be represented in Python.

Documentation for sets in Python can be found here: [Sets](#)

(<https://docs.python.org/3.6/library/stdtypes.html#set-types-set-frozenset>)

Sets and Set Operations: A Summative Example

To put this all together, let's consider an example with restaurants.

Example Setup

Think about a set A with all the restaurants that serve Italian food.

Next, there is a set B with all the restaurants that serve burgers.

You could say that the **universal set** here, set U , contains all the restaurants in the world (with any type of food).

Implications

The **union** of these sets, set C , contains the set of restaurants that either serve Italian food, burgers or both. Then A and B are both **subsets** of C .

The **intersection** of A and B contains the restaurants that *serve both Italian food and burgers*.

The **cardinality** of C (the union of A and B), is the number of restaurants that serve Italian food plus the number of restaurants that server burgers minus the intersection (restaurants that serve both Italian food and burgers).

The **relative complement** of A in B contains the restaurants that *do serve burgers but do not serve Italian food*.

The **absolute complement** of A contains the restaurants that *do not serve Italian food* (regardless of whether or not they serve burgers).

In this section, you learned about sets, subsets, and universal sets. Next, you were introduced to some core set operations such as unions, intersections, and complements. After that, all this information was tied together through the inclusion-exclusion principle and a summative example. You also saw how sets translate into Python. You'll start exploring this in further detail in the next lab!

How do you feel about this lesson?



Have specific feedback?

[Tell us here! \(https://github.com/learn-co-curriculum/dsc-intro-to-sets/issues/new/choose\)](https://github.com/learn-co-curriculum/dsc-intro-to-sets/issues/new/choose)