Discrete Structures: CMPSC 102

Oliver BONHAM-CARTER

Spring 2024
Week 9

Let's Discuss

Key Questions

How do I implement finite sets in Python so that I can calculate and use probabilities?

Learning Objectives

To **remember** and **understand** some concepts about **sets**, as implemented by `SymPy`, supporting the calculation of probabilities.

Mathematical Sets in Python Programs

- Set theory is useful in mathematics and computer science
- The `Sympy` package gives an implementation of finite sets
  - Remember, sets are "containers" for other elements
  - The sets in **Sympy** are finite sets, called **FiniteSet**
  - These sets have the same properties as built-in sets
  - **FiniteSet** has a few features not provided by **set**
  - A probability is the likelihood that an event will occur
  - We can use either **set** or **FiniteSet** to study probabilities
- Investigate probability after exploring an alternative approach to sets

Setting Up Virtual Environment

Create a project directory (outside of `lassDocs/`)

```
mkdir projects
cd projects
```

Create virtual environment using Python

```
python3 -m venv myenv
# see the file tree
find . -not -path  '*/\.*'    # UNIX only
dir /s    # Windows?
```

Activate *myenv* the virtual environment

```
source myenv/bin/activate
```

Install Dependencies

```
pip install sympy
```

Creating Sets

Import sympy

Get into a Python instance from terminal

```
python3
```

Creating a finite set

```
#_________________________
import sympy as sy
#
empty_set = sy.FiniteSet()
print(f"{empty_set} :: {type(empty_set)}")
# EmptySet :: <class 'sympy.sets.sets.EmptySet'>
```

Creating a finite set

```
import sympy as sy

finite_set = sy.FiniteSet(2, 4, 6, 8, 10)
print(f"{finite_set} :: {type(empty_set)}")
# <class 'sympy.sets.sets.EmptySet'>
```

Creating Sets

Creating a Set from a List or Tuple

```
#_________________________
import sympy as sy
#
mylist = [2, 4, 6, 8, 10]
finite_set = sy.FiniteSet(*mylist)
print(finite_set)
#
tuple = (2, 4, 6, 8, 10)
finite_set = sy.FiniteSet(*tuple)
print(finite_set)
```

- All approaches call the **FiniteSet** constructor
- Can construct a **FiniteSet** out of a list or a tuple
- What is the purpose of the "*" in this program?

The purpose of using "*" in set()

Part 1

We use the "*" operator for iterable unpacking. For instance, it is used to unpack the elements of the `list (mylist)` and pass them as separate arguments to the `sy.FiniteSet` constructor.

```
import sympy as sy
mylist = [1, 2, 3, 4, 5]
```

```
# Without *, crashes
finite_set_without_star = sy.FiniteSet(mylist)
print(f"Without *: {finite_set_without_star}")
```

The purpose of using "*" in set()

Part 2

```
# With *
finite_set_with_star = sy.FiniteSet(*mylist)
print(f"With *: {finite_set_with_star}")
```

- `sy.FiniteSet(*mylist)` is equivalent to explicitly writing `sy.FiniteSet(1, 2, 3, 4, 5)`
- The `*` operator unpacks the elements from the list and provides them as separate arguments to the function or constructor.
- Useful when you have a list of values that you want to pass individually to a function or constructor that expects multiple arguments.

Understanding Outputs

Output of Finite Set Creation Program

```
#_________________________
import sympy as sy
#
# Explicit FiniteSet:
sy.FiniteSet(2, 4, 6, 8, 10)
#
# Empty FiniteSet:
EmptySet
#
# FiniteSet from Tuple:
sy.FiniteSet(2, 4, 6, 8, 10)
#
# FiniteSet Containing Tuple:
sy.FiniteSet((2, 4, 6, 8, 10))
```

Why do we need this dependency?

Math and Programming Differences

- Programmers **cannot** use sets like mathematicians do!
- Python programs **cannot** store an infinite set
- Finite sets must **fit** into a computer's **finite** memory
- Programs need a **procedure** for **constructing** the set
- Different programming languages and packages have other restrictions. For instance, recall that Python programs **cannot** create sets that **contain mutable elements** like lists! Why do you think that this is the case?
- So, what are the **benefits** of using sets in Python programs?
- Importantly, sets come with some **super-useful** default operations!
- Thankfully, `sympy` contains even more basic operations!

Creating Sets

Using Finite Sets in Sympy

```
#_________________________
from sympy import FiniteSet
#
list = [1, 2, 3, 2]
finite_set = FiniteSet(*list)
print(finite_set)
#
for element in finite_set:
    print(element)
```

- What is the output of `print(finite_set)` ?
- What is the output of `print(element)` in the `for` loop?
- How do these two output segments differ?

Creating Sets

Subset Relationships with Finite Sets

```
#_________________________
from sympy import FiniteSet
#
one = FiniteSet(1, 2, 3)
two = FiniteSet(1, 2, 3)
#
subset = one.is_proper_subset(two)
print(subset)
subset = two.is_proper_subset(one)
print(subset)
```

- What is the mathematical definition of a **proper subset**?
- What is the purpose of the `is_proper_subset` function?
- What is the output of the `print(subset)` function calls?

Creating Sets

Subsets with Finite Sets

```
#_________________________
from sympy import FiniteSet
one = FiniteSet(1, 2, 3)
three = FiniteSet(1, 2, 3, 4)
#
subset = one.is_proper_subset(three)
print(subset)
subset = three.is_proper_subset(one)
print(subset)
```

- Is one a proper subset of three ?
- Is three a proper subset of one ?
- What is the output of the print(subset) ?

Determining Subsets with Sympy

```
#_________________________
from sympy import FiniteSet
one = FiniteSet(1, 2, 3)
two = FiniteSet(1, 2, 3)
three = FiniteSet(1, 2, 3, 4)
#
# Set one proper subset set two:
one.is_proper_subset(two) # False
#
# Set two proper subset set one:
two.is_proper_subset(one) # False
#
# Set one proper subset set three:.
one.is_proper_subset(three) # True
#
# Set three proper subset set one:
three.is_proper_subset(one) # False
```

Creating Sets

Union and Intersection with Finite Sets

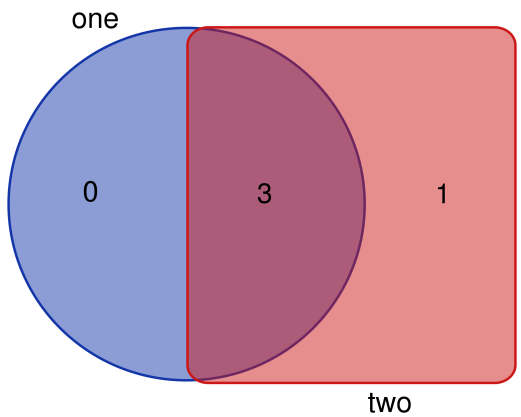
```
#_________________________
from sympy import FiniteSet
one = FiniteSet(1, 2, 3)
two = FiniteSet(1, 2, 3, 4)
#
intersection = one.intersection(two)
print(intersection)
#
union = one.union(two)
print(union)
```

- What is the meaning of `one.union(two)` ?
- What is the meaning of `one.intersection(two)` ?

As a Venn Diagram

Union and Intersection with Finite Sets

```
#_________________________
one = FiniteSet(1, 2, 3)
two = FiniteSet(1, 2, 3, 4)
#
intersection = one.intersection(two)
print(len(intersection)) # 3
#
union = one.union(two)
print(len(union)) # 4
```



Probability

Intersection

A die can roll prime numbers ($\{2, 3, 5\}$) or odd numbers ($\{1, 3, 5\}$). What are the chances of a die roll is both prime **AND** odd? To determine this, you calculate the probability of the **intersection** of the two event sets. $E = A \bigcap B = \{2, 3, 5\} \bigcap \{1, 3, 5\} = \{3, 5\}$

Probability of Event A AND Event B

```
#_________________________
six_sided = FiniteSet(1, 2, 3, 4, 5, 6)
roll_one = FiniteSet(2, 3, 5)
roll_two = FiniteSet(1, 3, 5)
event = roll_one.intersect(roll_two)
prob = len(event) / len(six_sided)
print(prob)
```

- The 'intersect' function connects to a logical 'AND' operation
- The output of this program is `0.3333333333333333`. Why?

Probability

Union

A die can roll prime numbers ($\{2, 3, 5\}$) or odd numbers ($\{1, 3, 5\}$). What are the chances of a die roll is both prime **OR** odd? To determine this, you calculate the probability of the **union** of the two event sets.
$E = A \bigcup B = \{2, 3, 5\} \bigcup \{1, 3, 5\} = \{1, 2, 3, 5\}$

Probability of Event A **OR** Event B

```
#_________________________
six_sided = FiniteSet(1, 2, 3, 4, 5, 6)
roll_one = FiniteSet(2, 3, 5)
roll_two = FiniteSet(1, 3, 5)
event = roll_one.union(roll_two)
prob = len(event) / len(six_sided)
print(prob)
```

- The 'union' function connects to a logical 'OR' operation
- The output of this program is `0.6666666666666666`. Why?

Creating Solutions

Check out your `sandbox`!