IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

Understanding Sets with the help of Python

[1] Yashasvini Raghuvanshi,[2]Dhananjay R. Mishra,[3]Pankaj Dumka

[1]B. Tech. Student,[2]Assistant Professor (SG),[3] Assistant Professor (SG)
[1]Department of Computer Science and Engineering,
[2,3]Department of Mechanical Engineering,
[1,2,3] Jaypee University of Engineering and Technology, A.B. Road, Raghogarh-473226, Guna, Madhya Pradesh (India)

Abstract : In this manuscript, basic set theory has been reviewed using Python programming language. SymPy has been used to develop sets, and different set operations have also been performed with the help of Python. Moreover, the set operations have also been explained diagrammatically with the help of Veen diagrams. This article will help a newcomer in the field of mathematics to implement, view, and work with complex set problems with the use of the methodology mentioned in this manuscript.

IndexTerms **- Set theory, SymPy, Venn diagrams, Python Programming.**

I. INTRODUCTION

Set theory is the study of correct rationality. A set is a collection of similar types of elements[1]. The concept of set theory is self-describing, and it is believed that all abstract mathematical concepts can be explained and constructed through it[2]. The prominent theory of sets is mostly dealing with the countable set. Still, also it arose due to uncountable entities and marked that there are infinite natural numbers and real numbers[3]. Therefore, it provides us with a better understanding of infinite objects and better sorting and solving of similar finite objects. In business operations and scheduling, concepts of the set theory are vividly implemented. The logical relation of sets is depicted through Venn diagrams[4]. A Venn diagram is a logic or set diagram of mathematical objects represented through circles. If the circles in the Venn diagram overlap, it is used to convey various set operations like intersection, union, and difference. The symbolic visualization set data is shown by Venn diagrams[5].
Here comes the importance of the SymPy module[6–10] and matplotlib[11–16] in the python language. SymPy is one of the python libraries for symbolic mathematical calculations. This module aims to keep the code as simple as possible while targeting the computer algebra system. Matplotlib is another library of Python dealing with graph plotting and data visualization. As a set is a collection of unique but similar entities, it is beneficial to perform operations and calculate through python language as it will easily identify the duplicate entities and remove them without distorting the whole objective.

II. SETS AND THEIR PYTHON IMPLEMENTATION

A set is an accumulation of unorganized items. These items are enclosed in curly brackets and have a unique value. There can be a set of people, numbers, places, names, etc., and each item from a set of people and respective sets will be called an object or element. For example:

colors = {purple, black, blue, green,"brown"}

Here, the set's name is colors, and its elements are pink, blue, navy blue, yellow, and orange. Quick observations from the above example are that all elements are unique and are enclosed in curly brackets and as the data type is string the input is in quotes. Below is the code which creates the set colors, and this has been done with the help of the FiniteSet[17] function, which will be explained in section 2.1.

Code	Output

`from sympy import FiniteSet`
`colors = FiniteSet("purple", "black","blue", "green" ,"brown")`
`print(colors)`
	`FiniteSet(black, blue, brown, green, purple)`

1.1 Sets using SymPy

SymPy is one of Python's libraries that provide modules for calculation and computations involving symbols used in mathematics. SymPy also provides a set module to perform various operations like union, intersection, complement, difference and many more. We can create sets of different kinds of numbers like integers, fractions, decimals, rational numbers, irrational numbers, etc., as shown below:

Code	Output

`from fractions import Fraction`
`a = FiniteSet(Fraction(7,11),5,3.5)`
`print(a)`
	`FiniteSet(7/11, 3.5, 5)`

Construction of a finite set: A finite set refers to a set of similar entities which can be counted and are non-repeated. In the SymPy module, there is already easy to class, class is a code template. One can check the cardinality of a set, cardinality means the number of elements in a certain set.

Code	Output

`from sympy import FiniteSet`
`w = FiniteSet(9,4,20,15,9,13,17)`
`print(w)`
`# for printing cardinality of set w`
`print ("length = ",len(w))`
	`FiniteSet(4, 9, 13, 15, 17, 20)`
	`length =  6`

Checking whether a number is there in a set or not: One can check whether the number that the user will use is there in the set by simply using the 'in' keyword. If the number is present in the mentioned set, the output will be true; else, the outcome will be false. The example shown below demonstrates the methodology.

Code	Output

`from sympy import FiniteSet`
`s = FiniteSet(1,5,9,8,6)`
`print( "2 in s : ",2in s )`
`print("8 in s : ",8in s)`
	`2 in s :  False`
	`8 in s :  True`

Empty set: As there is an empty set created in the mathematics notebook same can be created in Python. It is the set with no elements, as shown below:

Code	Output

`from sympy import FiniteSet`
`#empty set`
`es = FiniteSet()`
`print(es)`
	`EmptySet`

Using tuples and list form set: A list can be converted to a finite set by indirectly passing the list to the finite-set class using python syntax. The difference that will be noticed is that a set is enclosed in {} whereas a list is held in (). This is because set doesn't allow repeated elements in it. In contrast, a list and a tuple can have repeated elements in it, so when we convert a list into a set, all repeated elements will be removed.

Code	Output

`from sympy import FiniteSet`
`tup=(10,20,30)`
`lst =[6,3,9]`
`lst_set = FiniteSet(*lst)`
`tup_set = FiniteSet(*tup)`
`print("Set formed from list :", lst_set)`
`print("Set formed from tuple :",tup_set)`
	`Set formed from list :`
	`FiniteSet(3, 6, 9)`
	`Set formed from tuple :`
	`FiniteSet(10, 20, 30)`

Verification of unique elements in the list converted set: The following code will verify whether the set has unique elements or not. The order of elements will not be perverse in code while printing each element of the set, this can be helpful to understand that python stores the elements of the set but doesn't store elements' order in the set.

Code	Output

`from sympy import FiniteSet`
`l = [5,6,3,7,8,2,5,2]`
`s = FiniteSet(*l)`
`print("Original list :\n", l)`
`print("Set formed :\n",s)`
`for i in s:`
`print(i,end=';')`
	`Original list :`
	`[5, 6, 3, 7, 8, 2, 5, 2]`
	`Set formed :`
	`FiniteSet(2, 3, 5, 6, 7, 8)`
	`2;3;5;6;7;8;`

Equivalent sets: If two or more sets have the same elements, then we can check it using the '==' operator, now if we consider two sets and their elements are the same, then code will return true or else false. The order of elements doesn't play a role. A checking whether the elements are present or not rather than whether elements are present in order or not.

Code	Output

`from fractions import Fraction`
`s1 = FiniteSet(4,7.9,Fraction (1,2))`
`s2 = FiniteSet(4,7.9,Fraction (1,2))`
`s3 = FiniteSet(7.9,Fraction (1,2),4)`
`s4 = FiniteSet(7.9,Fraction (1,2))`
`print ("s1==s2: ",s1==s2)`
`print ("s1 == s2 == s3: ",s1 == s2 == s3)`
`print ("s1 == s4: ",s1 == s4)`
	`s1==s2: True`
	`s1 == s2 == s3: True`
	`s1 == s4: False`

Subset: Set A is said to be a subset of set B when all elements of A are present in B. The left set of elements should be present in the right set of elements to call a subset, but if the left set of elements is empty, then is it a subset of all sets. For example:

C= { }, A = {1,2,3}, B = {1,2,3,4}

$A \subset B, C \subset A$

Code	Output

`from sympy import FiniteSet`
`A = FiniteSet(1,2,3)`
`print("Set A: ",A)`
`B= FiniteSet(1,2,3,4)`
`print("Set B: ",B)`
`C= FiniteSet()`
`print("Set C: ",C)`
`print("A.is_subset(B): ",A.is_subset(B))`
`print("B.is_subset(A): ",B.is_subset(A))`
`print("C.is_subset(A): ",C.is_subset(A))`
`print("A.is_subset(C): ",A.is_subset(C))`
`print("C.is_subset(B): ",C.is_subset(B))`
	`Set A: FiniteSet(1, 2, 3)`
	`Set B: FiniteSet(1, 2, 3, 4)`
	`Set C: EmptySet`
	`A.is_subset(B): True`
	`B.is_subset(A): False`
	`C.is_subset(A): True`
	`A.is_subset(C): False`
	`C.is_subset(B): True`

Superset: A superset is a set when set elements of the right set are present in the left set. Considering the above sets, A, B, and C, if we say set A is contained in set B, B is a superset of A. For example:

C= { }, A = {1,2,3}, B = {1,2,3,4}

$B \supset A$

Code	Output

`from sympy import FiniteSet`
`A= FiniteSet(1,2,3)`
`B= FiniteSet(1,2,3,4)`
`C= FiniteSet()`
`print ("A.is_superset(B): ", A.is_superset(B))`
`print ("B.is_superset(A): ",B.is_superset(A))`
`print ("A.is_superset(C): ",A.is_superset(C))`
`print ("C.is_superset(A): ",C.is_superset(A))`
	`A.is_superset(B): False`
	`B.is_superset(A): True`
	`A.is_superset(C): True`
	`C.is_superset(A): False`

Power set: A power set is a set that contains all possible combinations of subsets with a null set. For example:

s = {1,2,3}

Power_set= { {},{1},{2},{3}, {1,2}, {2,3},{3,1},{1,2,3}}

Code	Output

`from sympy import FiniteSet`
`s= FiniteSet(1,2,3)`
`Power_set = A.powerset()`
`print("Length of Power Set", len(Power_set))`
`Power_set`
	`Length of Power Set 8`
	$\{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

Proper subset: The proper subset is a set with full elements of another set and elements of its own too. For example:

A = {1,2,3}
D = {1,2,3,4,5,6}
A is a proper subset of D

Code	Output

`from sympy import FiniteSet`
`A= FiniteSet(1,3,5)`
`D= FiniteSet(1,2,3,4,5,6)`
`print("A.is_proper_subset(D):",A.is_proper_subset(D))`
`print("D.is_proper_subset(A):",D.is_proper_subset(A))`
	`A.is_proper_subset(D): True`
	`D.is_proper_subset(A): False`

1.2 Set Operations

Union: By union operation, a new set is created with all the distinct elements from the selected sets. The selected set can be two or more. For example:

P = { 3 ,4,5,6}
Q= {6,4,8,9,10}
P U Q = {3,4,5,6,8,9,10}
We use the union () method for performing union operator in SymPy

Code	Output

`from sympy import FiniteSet`
`P= FiniteSet(3,4,5,6)`
`Q= FiniteSet(6,4,8,9,10)`
`R= FiniteSet(2,11,6,7,2)`
`print("P.union(Q):\n",P.union(Q))`
`print("P.union(Q).union(R):\n",P.union(Q).union(R))`
	`P.union(Q):`
	`FiniteSet(3, 4, 5, 6, 8, 9, 10)`
	`P.union(Q).union(R):`
	`FiniteSet(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)`

Intersection: The insertion operator generates a set that contains all the common elements of the set in which the insertion operator has been used, but all the elements are displayed once in the set. This operation can be performed in 2 or more sets. For example:

P = {3,4,5,6}
Q = {6,4,8,9,10}
R = {2,11,67,2}
Q ∩ R ={}
P ∩ Q = {4,6}
We use intersect () method for performing intersect operator in SymPy.

Code	Output

`from sympy import FiniteSet`
`P= FiniteSet(3,4,5,6)`
`Q= FiniteSet(6,4,8,9,10)`
`R= FiniteSet(2,11,6,7,2)`
`print("P.intersect(Q):\n",P.intersect(Q))`
`print("P.intersect(Q).intersect(R):\n",P.intersect(Q).intersect(R))`
	`P.intersect(Q):`
	`FiniteSet(4, 6)`
	`P.intersect(Q).intersect(R):`
	`FiniteSet(6)`

Cartesian product: A new set is generated from selected sets that contains ordered pairs of each element from the selected set. For example:

A = {1,2}
B = {8,9}
A X B = {(1,8), (1,9),(2,8),(2,9)}
There can be a Cartesian product of a set from itself

Code	Output

`from sympy import FiniteSet`
`A=FiniteSet(1,2)`
`B=FiniteSet(8,9)`
`z= A*B`
`print(z)`
`print('elements are:')`
`for e in z:`
`print(e)`
`print('Checking the cardinality of the`
	`ProductSet(FiniteSet(1, 2), FiniteSet(8, 9))`
	`elements are:`
	`(1, 8)`
	`(2, 8)`
	`(1, 9)`
	`(2, 9)`
	`Checking the cardinality of the Cartesian product z =A X B`
	`Does the len(z)=len(A)*len(B): True`

```
Cartesian product z =A X
B')
# Check for lenght
print("Does the
len(z)=len(A)*len(B):
",(len(z)==len(A)*len(B)))
```

1.3 Application of sets in the formula [17]

The time period of the pendulum to complete one full swing is to be evaluated with a set of different lengths for the pendulum. In this calculation, multiple sets of variables will be used to perform for correct output. The formula for the time is $T = 2\pi\sqrt{\frac{L}{g}}$. Where, g = 9.8 m/s^{2}and pi =3.14. Let the length in cm be formed as a set as: L = {10.76, 28, 20, 25.5, 15}. Then the code for the evaluation of the time be as follows:

Code

```
from sympy import*
def time_period(length):
  g =9.8
  T =2*pi*(length/g)**0.5
return T

L = FiniteSet(10.76, 28, 20, 25.5, 15)
print(f"{'Length(cm)':15} {'Time
Period(s)':15}")
print("------------------------------")
for i in L:
  t = time_period(i/100)
print(f'{float(i):5.2f}  \t
{float(t):17.3f}')
```

Output

```
Length(cm)       Time Period(s)
------------------------------
10.76                    0.658
15.00                    0.777
20.00                    0.898
25.50                    1.014
28.00                    1.062
```

We consider slight changes in the force of gravity at different locations on earth. So now, if we again calculate the time period for the pendulum with the same lengths as above in centimeters using the cartesian product, the result will following

Code

```
from sympy import FiniteSet, pi
def time_period(length, g):
  T =2*pi*(length/g)**0.5
return T

L = FiniteSet(10.76, 28, 20, 25.5,
15)
g_values = FiniteSet(9.81, 9.9,
9.79)
print(f"{'Length(cm)':^15}{
'Gravity(m/s^2)':^15}{'Time
Period(s)':^15}")
for i in L*g_values:
  l = i[0]
  g = i[1]
  t = time_period(l/100, g)
print(f'{float(l):^15}{float(g):^15}
{float(t):^15.3}')
```

Output

```
  Length(cm)    Gravity(m/s^2)  Time Period(s)
-----------------------------------------------
      10.76          9.79            0.659
      15.0           9.79            0.778
      10.76          9.81            0.658
      20.0           9.79            0.898
      15.0           9.81            0.777
      10.76           9.9            0.655
      25.5           9.79            1.01
      20.0           9.81            0.897
      15.0            9.9            0.773
      28.0           9.79            1.06
      25.5           9.81            1.01
      20.0            9.9            0.893
      28.0           9.81            1.06
      25.5            9.9            1.01
      28.0            9.9            1.06
```

1.4 Venn diagram

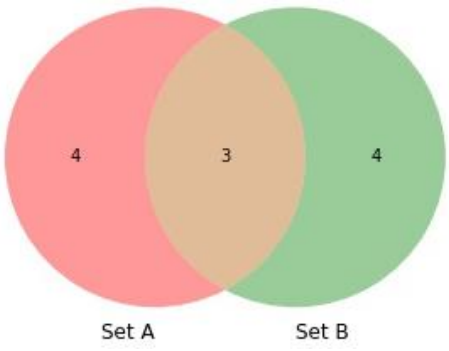
Venn diagrams are a graphical representation of sets and their relationship with another set in the universal set. The Venn diagram depicts the common and unique elements of sets in relation. The function venn2() is used to draw Venn diagram [17]. Let us take an example to understand how the Venn diagrams are drawn using Python

A = {2,3,4,5,9,6,11}

B= {2,3,5,71,1,44,43}

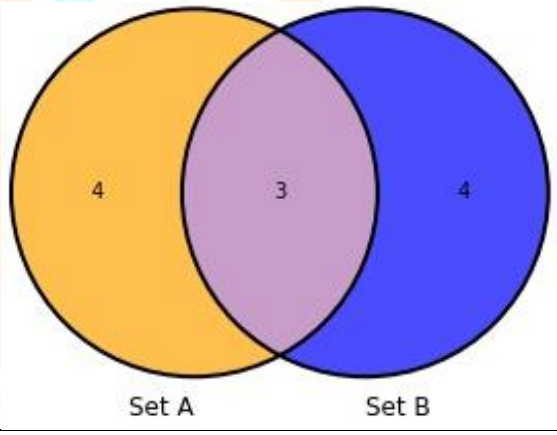
We need to import the libraries for executing the operation.

Code	Output

`from sympy import FiniteSet`
`from matplotlib.pyplot import*`
`from matplotlib_venn import venn2`
`def draw_venn(sets):`
`  venn2(subsets=sets,set_labels = ['Set A', 'Set B'])`
 	
`  figure(1,dpi=300)`
`  savefig('ven.jpg')`
`  show()`
 	
`A = FiniteSet(2,3,4,5,9,6,11)`
`B = FiniteSet(2,3,5,71,1,44,43)`
`draw_venn([A,B])`
	
	The Venn diagram is labeled with a few numbers like 4 and 3; they are the number of elements that are common and unique in sets A and B.

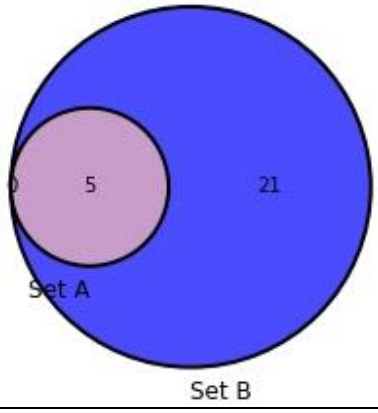
If the default colors are to be replaced by the choice of user then one can set the color by using parameter '`set_colors`'(eg: `set_colors=("orange","blue"),alpha=0.7)`. We can customize the circle's outline using the '`venn2_circles`'functions. The program snippet for the same is shown below:

Code	Output

`from sympy import FiniteSet`
`from matplotlib.pyplot import*`
`from matplotlib_venn import venn2,venn2_circles`
`def draw_venn(sets):`
`  venn2(subsets=sets,set_labels = ['Set A', 'Set B'],set_colors=("orange", "blue"),alpha=0.7)`
`  venn2_circles(sets)`
`  figure(1,dpi=300)`
`  savefig('ven.jpg')`
`  show()`
 	
`A = FiniteSet(2,3,4,5,9,6,11)`
`B = FiniteSet(2,3,5,71,1,44,43)`
`draw_venn([A,B])`
	

Let's say that A and B are two sets of vowels and all the alphabets in English then one cannot use FiniteSet instead set() function has to be used as shown below:

Code	Output

`A =set(['a','e','i','o','u'])`
`B =set(['a','b','c','d','e','f', 'g','h','i','j','k','l','m'`
`,'n','o','p','q','r','s','t', 'u','v','w','x','y','z'])`
 	
`draw_venn([A,B])`
	

III. Conclusion

In this research article, an attempt has been made to acknowledge the set theory basics and its implementation in python programming. The fundamental of sets, construction of sets, empty set, subset, superset, power set, proper set, and set operations like union, intersection, cartesian product, and Venn diagrams, along with codes for printing the correct output, has been developed. The article will be of great help to undergraduate students in understanding the set theory principle along with its programming aspects.

REFERENCES

[1] Jech TJ, Jech T, Jech TJ, Mathematician GB, Jech TJ, Mathématicien G-B. Set theory. vol. 14. Springer; 2003.
[2] Hausdorff F. Set theory. vol. 119. American Mathematical Soc.; 2005.
[3] Fraenkel AA, Bar-Hillel Y, Levy A. Foundations of set theory. Elsevier; 1973.
[4] Alsina C, Nelsen R. Venn Diagrams. Icons Math 2011:149–62. doi:10.5948/upo9780883859865.014.
[5] Gunstone RF, White RT. Assessing understanding by means of Venn diagrams. Sci Educ 1986;70:151–8.
[6] Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, et al. SymPy: Symbolic computing in python. PeerJ Comput Sci 2017;2017:1–27. doi:10.7717/peerj-cs.103.
[7] Cywiak M, Cywiak D. SymPy. Multi-Platform Graph. Program. with Kivy Basic Anal. Program. 2D, 3D, Stereosc. Des., Berkeley, CA: Apress; 2021, p. 173–90. doi:10.1007/978-1-4842-7113-1_11.
[8] Pawar PS, Mishra DR, Dumka P, Pradesh M. OBTAINING EXACT SOLUTIONS OF VISCO- INCOMPRESSIBLE PARALLEL FLOWS USING PYTHON. Int J Eng Appl Sci Technol 2022;6:213–7.
[9] Dumka P, Chauhan R, Singh A, Singh G, Mishra D. Implementation of Buckingham ' s Pi theorem using Python. Adv Eng Softw 2022;173:103232. doi:10.1016/j.advengsoft.2022.103232.
[10] Pawar PS, Mishra DR, Dumka P. Solving First Order Ordinary Differential Equations using Least Square Method : A comparative study. Int J Innov Sci Res Technol 2022;7:857–64.
[11] Dumka P, Deo A, Gajula K, Sharma V, Chauhan R, Mishra DR. Load and Load Duration Curves Using Python. Int J All Res Educ Sci Methods 2022;10:2127–34.
[12] Gajula K, Sharma V, Sharma B, Mishra DR, Dumka P. Modelling of Energy in Transit Using Python. Int J Innov Sci Res Technol 2022;7:1152–6.
[13] Dumka P, Rana K, Pratap S, Tomar S, Pawar PS, Mishra DR. Modelling air standard thermodynamic cycles using python. Adv Eng Softw 2022;172:103186. doi:10.1016/j.advengsoft.2022.103186.
[14] Bisong E. Matplotlib and Seaborn. Build. Mach. Learn. Deep Learn. Model. Google Cloud Platf., Berkeley, CA: Apress; 2019, p. 151–65. doi:10.1007/978-1-4842-4470-8_12.
[15] Porcu V. Matplotlib. Python Data Min. Quick Syntax Ref., Berkeley, CA: Apress; 2018, p. 201–34. doi:10.1007/978-1-4842-4113-4_10.
[16] Johansson R. Numerical python: Scientific computing and data science applications with numpy, SciPy and matplotlib, Second edition. Apress, Berkeley, CA; 2018. doi:10.1007/978-1-4842-4246-9.
[17] Saha A. Doing Math with Python: Use Programming to Explore Algebra, Statistics, Calculus, and More! No Starch Press; 2015.