Module 2: Mathematical Logic

Propositional Logic

A proposition is a sentence that declares a fact that is either True or False. In Python, we can use boolean variables (typically p and q) to represent propositions and define functions for each propositional rule. Each rule can be implemented using the boolean operators (and, or, not, etc).

Logical Expressions in Sympy

SymPy is a symbolic mathematics Python package. Its goal is to develop into a completely featured computer algebra system while keeping the code as basic as possible to make it understandable and extendable. The package is entirely written in python language. Logical expressions in sympy are expressed by using boolean functions. **sympy.basic.booleanarg** module of sympy contains boolean functions.

The common Python operators **& (And), | (Or),** and **~ (Not)** can be used to create Boolean expressions. **>>(Implies)** and **(Equivalent)** can be used for to create implication and bi-implication. Other boolean operations or gates are **NAND, NOR, XOR, etc**.

Topics :

1. Negation
2. Conjunction
3. Disjunction
4. Exclusive Disjunction
5. Implification
6. Bi-implification
7. Python program for the compound proposition
8. Rules of inference using Python

Negation

The negation is a statement that has the opposite truth value. The negation of a proposition p, denoted by ¬p, is the proposition "It is not the case , that p".

For example, the negation of the proposition "Today is Friday." would be "It is not the case that, today is Friday." or more succinctly "Today is not Friday".

```
from sympy.logic.boolalg import Not
def negation(p):
    return Not(p)
print("p    ans")
for p in [True, False]:
    ans = negation(p)
    print(p, ans)
```

```
 p    ans
True False
False True
```

Sample Python Program based on negation / NOT [~]

Python program to check whether the input number 10 is divisible by 5 or 3.

```
a = float(input("Enter the number: "))
if not (a%3==0 or a%5==0):
  print("10 is not divisible by either 3 or 5")
else:
   print("10 is divisible by either 3 or 5")

Enter first number: 10
10 is divisible by either 3 or 5
```

Conjunction

Let p and q be propositions. The conjunction of p and q, denoted in mathematics by pΛq, is True when both p and q are True, False otherwise.

sympy.logic.boolalg.And

It analyzes each of its arguments in sequence, it returns true if all of the arguments are true. if at least one argument is false, false is returned.

```
from sympy.logic.boolalg import And
def conjunction(p,q):
    return And(p,q)
print("p     q     ans")
for p in [True, False]:
  for q in [True, False]:
    ans = conjunction(p,q)
    print(p,q, ans)
```

```
p     q     ans
True True True
True False False
False True False
False False False
```

Sample Python Program based on Conjunction/ AND [∧]

Python program to find the largest number among the three input numbers.

```
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
num3 = float(input("Enter third number: "))
if (num1 >= num2) and (num1 >= num3):
   largest = num1
elif (num2 >= num1) and (num2 >= num3):
   largest = num2
else:
   largest = num3
print("The largest number is", largest)
```

```
Enter first number: 10
Enter second number: 12
Enter third number: 8
The largest number is 12.0
```

Disjunction

Let p and q be propositions. The disjunction of p and q, denoted in mathematics by pVq, is True when at least one of p and q are True, False otherwise.

sympy.logic.boolalg.Or

It analyzes each of its arguments in sequence, it returns true if any of the arguments is true otherwise the argument is false.

```
from sympy.logic.boolalg import Or
def disjunction(p,q):
    return Or(p,q)
print("p     q     ans")
for p in [True, False]:
  for q in [True, False]:
    ans = disjunction(p,q)
    print(p,q, ans)
```

```
p     q     ans
True True True
True False True
False True True
False False False
```

Sample Python Program based on Disjunction/ OR [V]

Python program to check whether all the two input numbers are positive.

```
a = float(input("Enter first number: "))
b = float(input("Enter second number: "))
if a>0 or b>0:
  print("Atleast one of the numbers is greater than zero")
else:
   print("No number is greater than zero")
```

```
Enter first number: 8
Enter second number: -10
Atleast one of the numbers is greater than zero
```

Exclusive Disjunction

Let p and q be propositions. The exclusive disjunction of p and q (also known as xor), denoted in mathematics by p⊕q, is True when exactly one of p and q are True, False otherwise.

```
from sympy.logic.boolalg import Or, And, Not
def exclusive_disjunction(p,q):
    return Or(And(p,Not(q)),And(Not(p),q))
print("p     q     ans")
for p in [True, False]:
  for q in [True, False]:
    ans = exclusive_disjunction(p,q)
    print(p,q, ans)
```

```
p     q     ans
True True False
True False True
False True True
False False False
```

Implication

Let p and q be propositions. The implication of p and q, denoted in mathematics by p⟹q, is short hand for the statement "if p then q".

sympy.logic.boolalg.Implies

In other words, implication fails (is False) when p is True and q is False otherwise the argument is True.

```
from sympy.logic.boolalg import Implies
def implication(p,q):
    return Implies(p,q)
print("p     q     ans")
for p in [True, False]:
  for q in [True, False]:
    ans = implication(p,q)
    print(p,q, ans)
```

```
p     q     ans
True True True
True False False
False True True
False False True
```

Bi-Implication

Let p and q be propositions. The bi-implication of p and q, denoted in mathematics by p⇔q, is short hand for the statement "p if and only if q".

sympy.logic.boolalg.Equivalent

As such, bi-implication requires q to be True only when p is True. In other words, bi-implication fails (is False) when p is True and q is False or when p is False and q is True.

```
from sympy.logic.boolalg import Equivalent
def bi_implication(p,q):
    return Equivalent(p,q)
print("p     q     ans")
for p in [True, False]:
  for q in [True, False]:
    ans = bi_implication(p,q)
    print(p,q, ans)
```

```
p     q     ans
True True True
True False False
False True False
False False True
```

Python program for the compound proposition

Construct the Python program for the followoing compound proposition

$[(p \wedge q) \vee \neg r] \leftrightarrow p$

```
from sympy.logic.boolalg import Or, And, Not, Equivalent
def compound_prop(p,q,r):
      return Equivalent(Or(And(p,q),Not(r)),p)
print("p     q      r   ans")
for p in [True, False]:
 for q in [True, False]:
  for r in [True, False]:
      ans=compound_prop(p,q,r)
      print(p,q,r, ans)
p     q      r   ans
True True True True
True True False True
True False True False
True False False True
False True True True
False True False False
False False True True
False False False False
```

Rules of Inference using Python

Test Whether following is a valid argument by using Python code:

If Sachin hits a century then he gets a free car

sachin does not get a free car.

∴Sachin has not hit century

p: Sachin hits a century, q: Sachin gets a free car

$p \rightarrow q$

$\sim q$

$\therefore \sim p$

We need to prove $[(p \rightarrow q) \wedge \sim q] \rightarrow \sim p$ is tautology

```
from sympy.logic.boolalg import Or, And, Not, Implies
def rules_of_inference(p,q):
    return Implies(And(Implies(p,q),Not(q)),Not(p))
print("p     q     ans")
for p in [True, False]:
  for q in [True, False]:
    ans = rules_of_inference(p,q)
    print(p,q, ans)
```

```
p     q     ans
True True True
True False True
False True True
False False True
```

Exercise question

1. Write the python code for the following compound proposition:

(i) $p \vee \neg q$

(ii) $p \rightarrow \neg q$

(iii) $p \rightarrow (p \vee q)$

(iv) $p \wedge (\neg p \wedge q)$

(v) $(p \vee q) \rightarrow r$

(v) $\{(p \vee q) \wedge [(p \rightarrow r) \wedge (q \rightarrow r)]\} \rightarrow r$

2. Test whether following is a valid argument or not by using python code

   If Sachin hits a century then he gets a free car

   Sachin hits a century

   ∴ sachin gets a free car

3. Test whether following is a valid argument or not by using python code

   If I Study, then I do not fail in the exam

   If I do not fail in the exam, then my father gifts a car to me

   ---

   ∴ If I study then my father gift me a car