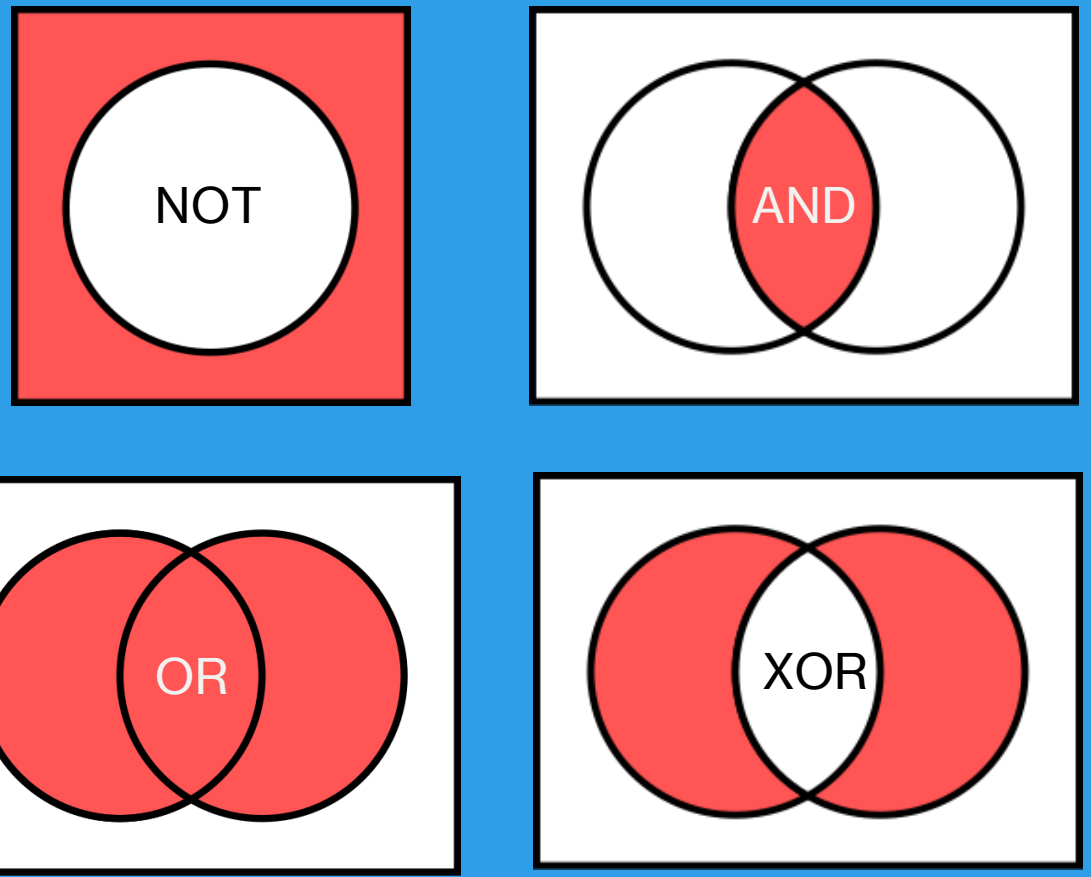
Computer Systems

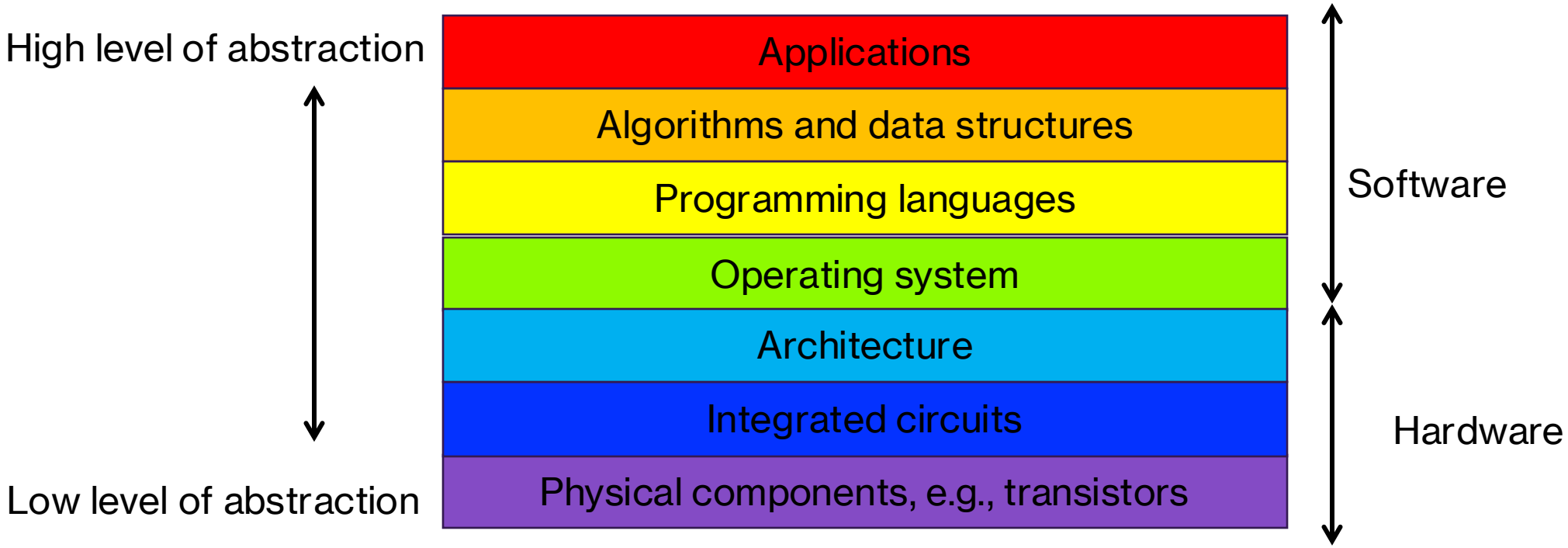


Boolean Algebra

CS51 – Fall 2025

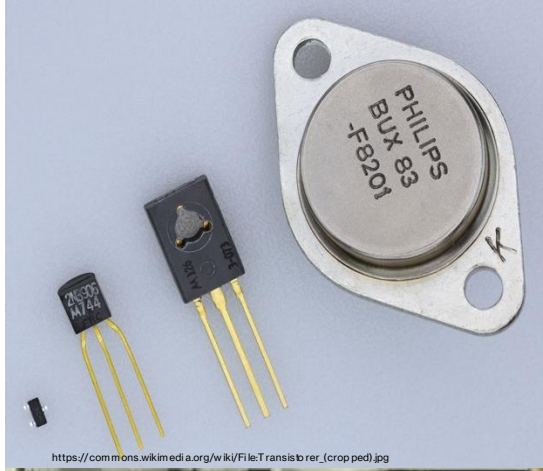
Abstraction

- Critical technique for managing complexity. We will simplify and generalize information by hiding details when they are not important. This allows us to cultivate a higher level of understanding without being overwhelmed with insignificant details.



Transistors

- Transistors are semiconductor devices made out of silicon.
- They act as switches that can be open or closed by applying electricity.
- Integrated circuits in modern computers are built off billions of transistors.
  - They are **small** – you can fit a few thousand of them across the width of a human hair!
  - They are **fast** – they can switch states millions of times per second.
  - They are **reliable** – they can run without any issue for decades.
- Invented in 1947 by Bardeen, Brattain, and Shockley at Bell Lab.
  - They shared the 1956 Nobel Prize in Physics for their achievement.
- A lot of development of transistors and circuits took place in Santa Clara Valley, which by the 1980s started being referred to as the **Silicon Valley**.



https://commons.wikimedia.org/wiki/File:Transistorer_(cropped).jpg





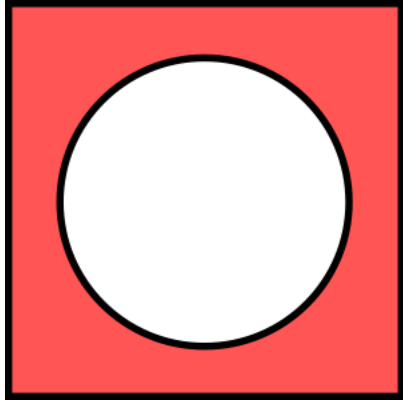
Binary representation and Boolean algebra

- Transistors are powered with electricity and can be in one of two states, allowing us to represent information in **binary**.
- Binary means two states. Common interchangeable representations are:
  - 0 and 1,
  - FALSE and TRUE,
  - LOW and HIGH.
- Working only with two states has the advantage of giving us a distinct signal, thus minimizing errors.
- It also works with a well-established branch of Mathematics called **Boolean algebra** that works with binary variables and logical operations.
  - George Boole was a self-taught English mathematician. He introduced **binary variables** and the logic operations **NOT, AND**, and **OR.**



George Boole

NOT operation – Negation



- Unary operator, i.e., it is applied to only one operand. Also denoted as ¬.
- It flips FALSE (0) to TRUE (1) and, equivalently, TRUE (1) to FALSE (0).
- We can use a **truth table** to enumerate all possible values of operands and the effect of the logical operations when applied to them.
- Truth table for NOT:

X	NOT X
FALSE	TRUE
TRUE	FALSE

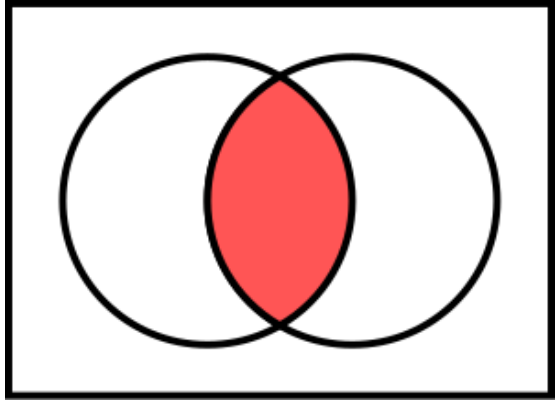
- Equivalently:

X	¬ X
0	1
1	0

- Logic gate (inverter) NOT:



AND operation – Conjunction



- Binary operator, i.e., it is applied two operands. Also denoted as ∧.
- Produces TRUE (1) only if *both* operands are TRUE (1).
- Truth table for AND:

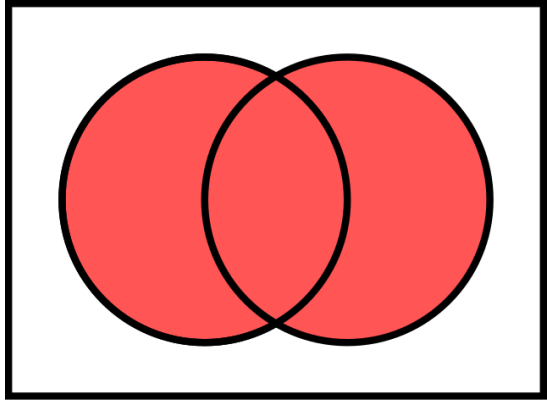
X	Y	X AND Y
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

- Equivalently:

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

- Logic gate AND: 

OR operation – Disjunction



- Binary operator, i.e., it is applied two operands. Also denoted as ∨.
- Produces TRUE (1) if *at least* one operand is TRUE (1).
- Truth table for OR:

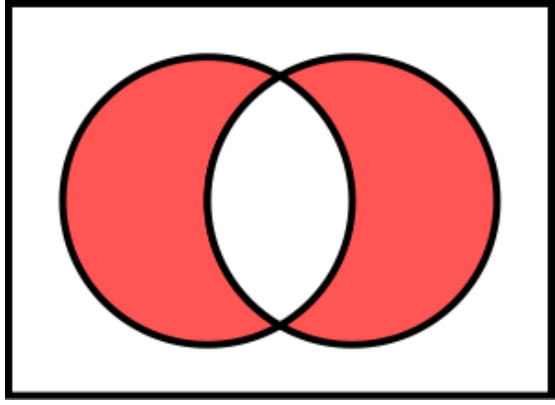
X	Y	X OR Y
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

- Equivalently:

X	Y	X ∨ Y
0	0	0
0	1	1
1	0	1
1	1	1

- Logic gate OR: 

XOR operation – Exclusive OR



- Binary operator, i.e., it is applied two operands. Also denoted as $\oplus$.
- Produces TRUE (1) if *exactly* one operand is TRUE (1).
- Truth table for XOR:

X	Y	X XOR Y
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

- Equivalently:

X	Y	X $\oplus$ Y
0	0	0
0	1	1
1	0	1
1	1	0

- Logic gate XOR:



PRACTICE TIME – De Morgan's laws

- $\neg (X \wedge Y) = \neg X \vee \neg Y$
- $\neg (X \vee Y) = \neg X \wedge \neg Y$

X	Y	$X \wedge Y$	$\neg (X \wedge Y)$	$X \vee Y$	$\neg (X \vee Y)$	$\neg X$	$\neg Y$	$\neg X \wedge \neg Y$	$\neg X \vee \neg Y$
0	0								
0	1								
1	0								
1	1								

ANSWER – De Morgan's laws

- ¬ (X ∧ Y) = ¬ X ∨ ¬ Y
- ¬ (X ∨ Y) = ¬ X ∧ ¬ Y

X	Y	X ∧ Y	¬ (X ∧ Y)	X ∨ Y	¬ (X ∨ Y)	¬ X	¬ Y	¬ X ∧ ¬ Y	¬ X ∨ ¬ Y
0	0	0	1	0	1	1	1	1	1
0	1	0	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	0	1
1	1	1	0	1	0	0	0	0	0

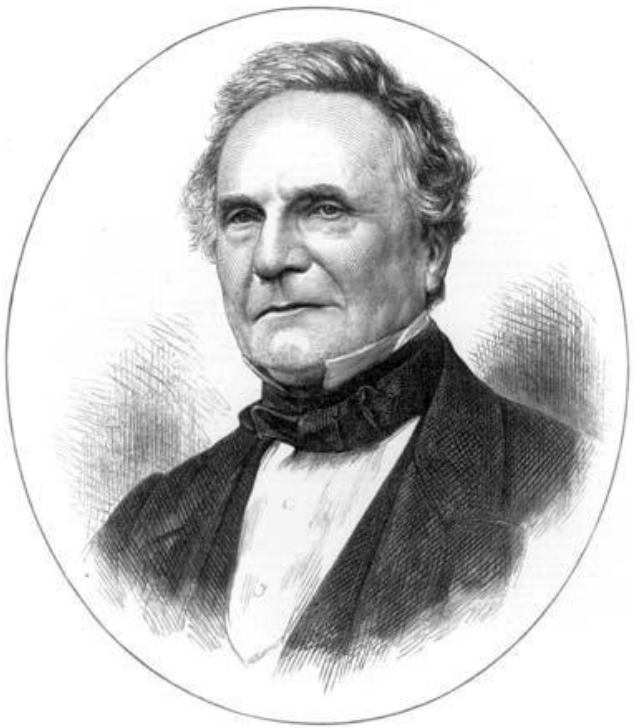
A mathematical lineage to the first computer



Augustus De Morgan



Ada Lovelace



Charles Babbage

PRACTICE TIME – Evaluate these expressions

- $0 \vee (\neg\ 0)$
- $1 \wedge (\neg\ 1)$
- $\neg\ (1 \wedge 0)$
- $0 \wedge (\neg\ 1)$
- $\neg\ (1 \vee (\neg\ 1))$
- $1 \oplus (\neg\ 1)$

ANSWER– Evaluate these expressions

- $0 \vee (\neg\ 0) = 0 \vee 1 = 1$
- $1 \wedge (\neg\ 1) = 1 \wedge 0 = 0$
- $\neg\ (1 \wedge 0) = \neg\ 0 = 1$
- $0 \wedge (\neg\ 1) = 0 \wedge 0 = 0$
- $\neg\ (1 \vee (\neg\ 1)) = \neg\ (1 \vee 0) = \neg\ 1 = 0$
- $1 \oplus (\neg\ 1) = 1 \oplus 0 = 1$

Boolean functions of three+ variables

- AND($X_1, X_2, \ldots, X_n$) = 0 if any argument is 0, 1 if all arguments are 1.
- OR($X_1, X_2, \ldots, X_n$) = 1 if any argument is 1, 0 if all arguments are 0.
- Infinite possibilities. Here are two examples of such Boolean functions:
  - MAJ($X_1, X_2, \ldots, X_n$) = 1 if (strictly) more arguments are 1 than 0, 0 otherwise.
  - ODD($X_1, X_2, \ldots, X_n$) = 1 if an odd number of arguments is 1, 0 otherwise.

X	Y	Z	AND(X,Y,Z)	OR(X,Y,Z)	MAJ(X,Y,Z)	ODD(X,Y,Z)
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	0	1
1	0	1	0	1	1	0
1	1	0	0	1	1	0
1	1	1	1	1	1	1

Disjunctive and conjunctive normal form

- **Theorem (Boole, 1847):** Any Boolean function can be represented as a disjunction (one OR) of conjunctions (a bunch of ANDs) of its arguments and their negation (NOT). This is also known as **disjunctive normal form** (DNF) or a sum of products/minterms.
- Equivalently, any Boolean function can be represented as a conjunction (one AND) of disjunctions (a bunch of ORs) of its arguments and their negation (NOT). This is also known as a **conjunctive normal form** (CNF) or product of sums/maxterms.
- How to find the disjunctive normal form of a function:
  - Using truth tables, we isolate the rows where the function evaluates to TRUE. Each row contributes a conjunction of the arguments (if TRUE) or their negation (if FALSE).
- How to find the conjunctive normal form of a function:
  - Using truth tables, we isolate the rows where the function evaluates to FALSE. Each row contributes a disjunction of the arguments (if FALSE) or their negation (if TRUE).

Minterms

Minterm: a Boolean expression consisting of the conjunction (AND) of all variables of a Boolean function. Each variable will appear exactly once in either its true (if the variable was 1) or complemented form (if the variable was 0). The minterm is formed using the values of the variables which make the value of the minterm equal to 1.

Row number	X	Y	Z	Minterm
0	0	0	0	$m_0 = \neg X \wedge \neg Y \wedge \neg Z$
1	0	0	1	$m_1 = \neg X \wedge \neg Y \wedge Z$
2	0	1	0	$m_2 = \neg X \wedge Y \wedge \neg Z$
3	0	1	1	$m_3 = \neg X \wedge Y \wedge Z$
4	1	0	0	$m_4 = X \wedge \neg Y \wedge \neg Z$
5	1	0	1	$m_5 = X \wedge \neg Y \wedge Z$
6	1	1	0	$m_6 = X \wedge Y \wedge \neg Z$
7	1	1	1	$m_7 = X \wedge Y \wedge Z$

Maxterms

Maxterm: a Boolean expression consisting of the disjunction (OR) of all variables of a Boolean function. Each variable will appear exactly once in either its true (if the variable was 0) or complemented form (if the variable was 1). The maxterm is formed using the values of the variables which make the value of the maxterm equal to 0.

Row number	X	Y	Z	Minterm	Maxterm
0	0	0	0	$m_0 = \neg X \wedge \neg Y \wedge \neg Z$	$M_0 = X \vee Y \vee Z$
1	0	0	1	$m_1 = \neg X \wedge \neg Y \wedge Z$	$M_1 = X \vee Y \vee \neg Z$
2	0	1	0	$m_2 = \neg X \wedge Y \wedge \neg Z$	$M_2 = X \vee \neg Y \vee Z$
3	0	1	1	$m_3 = \neg X \wedge Y \wedge Z$	$M_3 = X \vee \neg Y \vee \neg Z$
4	1	0	0	$m_4 = X \wedge \neg Y \wedge \neg Z$	$M_4 = \neg X \vee Y \vee Z$
5	1	0	1	$m_5 = X \wedge \neg Y \wedge Z$	$M_5 = \neg X \vee Y \vee \neg Z$
6	1	1	0	$m_6 = X \wedge Y \wedge \neg Z$	$M_6 = \neg X \vee \neg Y \vee Z$
7	1	1	1	$m_7 = X \wedge Y \wedge Z$	$M_7 = \neg X \vee \neg Y \vee \neg Z$

Example – DNF of MAJ function

- Remember, MAJ($X_1, X_2, \ldots, X_n$) = 1 if (strictly) more arguments are 1 than 0, 0 otherwise.

X	Y	Z	Minterm	Maxterm	MAJ(X,Y,Z)
0	0	0	$m_0 = \neg X \wedge \neg Y \wedge \neg Z$	$M_0 = X \vee Y \vee Z$	0
0	0	1	$m_1 = \neg X \wedge \neg Y \wedge Z$	$M_1 = X \vee Y \vee \neg Z$	0
0	1	0	$m_2 = \neg X \wedge Y \wedge \neg Z$	$M_2 = \mathrm{X} \vee \neg \mathrm{Y} \vee \mathrm{Z}$	0
0	**1**	**1**	$\mathbf{m_3 = \neg X \wedge Y \wedge Z}$	$M_3 = X \vee \neg Y \vee \neg Z$	**1**
1	0	0	$m_4 = X \wedge \neg Y \wedge \neg Z$	$M_4 = \neg X \vee Y \vee Z$	0
1	**0**	**1**	$\mathbf{m_5 = X \wedge \neg Y \wedge Z}$	$M_5 = \neg X \vee Y \vee \neg Z$	**1**
1	**1**	**0**	$\mathbf{m_6 = X \wedge Y \wedge \neg Z}$	$M_6 = \neg X \vee \neg Y \vee Z$	**1**
1	**1**	**1**	$\mathbf{m_7 = X \wedge Y \wedge Z}$	$M_7 = \neg X \vee \neg Y \vee \neg Z$	**1**

- To derive the disjunctive normal form of a function, we will isolate the rows where the output is equal to 1. We will then connect with a disjunction (OR) the corresponding minterms.
- For the MAJ Boolean function over three variables, this will result in:
  DNF of MAJ(X, Y, Z) = $m_3 \vee m_5 \vee m_6 \vee m_7 =$ (¬X ∧ Y ∧ Z) ∨ (X ∧ ¬Y ∧ Z) ∨ (X ∧ Y ∧ ¬Z) ∨ (X ∧ Y ∧ Z)

Example – CNF of MAJ function

- To derive the conjunctive normal form of a function, we will isolate the rows where the output is equal to 0. We will then connect with a conjunction (AND) the corresponding maxterms.

X	Y	Z	Minterm	Maxterm	MAJ(X,Y,Z)
0	0	0	$m_0 = \neg X \wedge \neg Y \wedge \neg Z$	$\boldsymbol{M_0 = X \vee Y \vee Z}$	**0**
0	0	1	$m_1 = \neg X \wedge \neg Y \wedge Z$	$\boldsymbol{M_1 = X \vee Y \vee \neg Z}$	**0**
0	1	0	$m_2 = \neg X \wedge Y \wedge \neg Z$	$\boldsymbol{M_2 = X \vee \neg Y \vee Z}$	**0**
0	1	1	$m_3 = \neg X \wedge Y \wedge Z$	$M_3 = X \vee \neg Y \vee \neg Z$	1
1	0	0	$m_4 = X \wedge \neg Y \wedge \neg Z$	$\boldsymbol{M_4 = \neg X \vee Y \vee Z}$	**0**
1	0	1	$m_5 = X \wedge \neg Y \wedge Z$	$M_5 = \neg X \vee Y \vee \neg Z$	1
1	1	0	$m_6 = X \wedge Y \wedge \neg Z$	$M_6 = \neg X \vee \neg Y \vee Z$	1
1	1	1	$m_7 = X \wedge Y \wedge Z$	$M_7 = \neg X \vee \neg Y \vee \neg Z$	1

- For the MAJ Boolean function over three variables, this will result in:
  CNF of MAJ(X, Y, Z) = $M_0 \wedge M_1 \wedge M_2 \wedge M_4$ = (X ∨ Y ∨ Z) ∧ (X ∨ Y ∨ ¬Z) ∧ (X ∨ ¬Y ∨ Z) ∧ (¬X ∨ Y ∨ Z)

PRACTICE TIME - ODD function

- Remember, ODD($X_1, X_2, \ldots, X_n$) = 1 if an odd number of arguments is 1, 0 otherwise.

X	Y	Z	Minterm	Maxterm	ODD(X,Y,Z)
0	0	0	$m_0 =$	$M_0 =$	0
0	0	1	$m_1 =$	$M_1 =$	1
0	1	0	$m_2 =$	$M_2 =$	1
0	1	1	$m_3 =$	$M_3 =$	0
1	0	0	$m_4 =$	M_4	1
1	0	1	$m_5 =$	$M_5 =$	0
1	1	0	$m_6 =$	$M_6 =$	0
1	1	1	$m_7 =$	$M_7 =$	1

- Disjunctive normal form (OR of minterms for rows that are 1) :

  ODD(X, Y, Z) =

- Conjunctive normal form (AND of maxterms for rows that are 0):

  ODD(X, Y, Z) =

ANSWER - ODD function

- Remember, ODD($X_1, X_2, \ldots, X_n$) = 1 if an odd number of arguments is 1, 0 otherwise.

X	Y	Z	Minterm	Maxterm	ODD(X,Y,Z)
0	0	0	$m_0 = \neg X \wedge \neg Y \wedge \neg Z$	$M_0 = X \vee Y \vee Z$	0
0	0	1	$m_1 = \neg X \wedge \neg Y \wedge Z$	$M_1 = X \vee Y \vee \neg Z$	1
0	1	0	$m_2 = \neg X \wedge Y \wedge \neg Z$	$M_2 = X \vee \neg Y \vee Z$	1
0	1	1	$m_3 = \neg X \wedge Y \wedge Z$	$M_3 = X \vee \neg Y \vee \neg Z$	0
1	0	0	$m_4 = X \wedge \neg Y \wedge \neg Z$	$M_4 = \neg X \vee Y \vee Z$	1
1	0	1	$m_5 = X \wedge \neg Y \wedge Z$	$M_5 = \neg X \vee Y \vee \neg Z$	0
1	1	0	$m_6 = X \wedge Y \wedge \neg Z$	$M_6 = \neg X \vee \neg Y \vee Z$	0
1	1	1	$m_7 = X \wedge Y \wedge Z$	$M_7 = \neg X \vee \neg Y \vee \neg Z$	1

- Disjunctive normal form (OR of minterms for rows that are 1) :
  ODD(X, Y, Z) = $m_1 \vee m_2 \vee m_4 \vee m_7$ = (¬X ∧ ¬Y ∧ Z) ∨ (¬X ∧ Y ∧ ¬Z) ∨ (X ∧ ¬Y ∧ ¬Z) ∨ (X ∧ Y ∧ Z)
- Conjunctive normal form (AND of maxterms for rows that are 0):
  ODD(X, Y, Z) = $M_0 \wedge M_3 \wedge M_5 \wedge M_6$ = (X ∨ Y ∨ Z) ∧ (X ∨ ¬Y ∨ ¬Z) ∧ (¬X ∨ Y ∨ ¬Z) ∧ (¬X ∨ ¬Y ∨ Z)

Boolean variables in Python

- In Python, Boolean variables are represented through the `bool` data type. A bool variable can either be `True` or `False`. These values can also be seen as equal to 1 and 0, respectively.
- For example, `raining = False` declares a variable called `raining` and assigns to it the value `False`.
- The comparison operators `==`, `!=`, `>`, `<`, `>=`, `<=` return bools. For example,
  - `10 < 0` evaluates to `False`
  - `11 >= 11` evaluates to `True`
  - `11 > 11.0` evaluates to `False`
  - `10 == 10.0` evaluates to `True`
  - `10 != 10.0` evaluates to `False`

Boolean Algebra in Python

- We can combine bools and logical operators in Boolean expressions, for example, to evaluate them in **conditionals** (`if` statements) and **loops** (`for` and `while`).
- Python supports three logical operators:
  - `not` for negation
  - `and` for conjunction
  - `or` for disjunction

Short-circuit evaluation in Python

- Python supports **short-circuiting**, a technique that evaluates a logical expression from left to right and stops once the truth can be determined.
  - When an expression with an `or` statement gets evaluated, it is enough to just see the first operand evaluate to `True` and return it. No need to check the second operand because `True or x` will evaluate to `True` no matter what `x` evaluates to.
  - When an expression with an `and` statement gets evaluated, it is enough to just see the first operand evaluate to `False` and return it. No need to check the second operand because `False and x`, will evaluate to `False` no matter what `x` evaluates to.

Short-circuit evaluation and optimization

- Because of short-circuit evaluation, we can avoid deeply nested if statements to handle exceptional cases. For example,
  - if x == 0 or (x-1)/x > 0.5: …
  - When x is equal to 0, evaluating the second operand of the or statement would cause a divide-by-zero error. But the second disjunct isn't evaluated when x is equal to 0 because the first operand was True!
- We can take advantage of short-circuit evaluation to make our code a bit faster. For example, when writing an if statement that contains the logical operator and, you should first write what is simpler to calculate or often false before something that is complex to calculate or often true:
  - if (simpleOrOftenFalse() and complexOrOftenTrue()):…

PRACTICE TIME – Boolean expressions

- Assume the variable x currently stores the value 47. What will the following expressions evaluate to?
  - True and not False
  - not True or not False
  - True and not 0
  - False or not 1
  - 2 < x and x < 50
  - x > 2 and < 20
  - (2 > x) or (x == 47)
  - not (x == x)
  - not (x != 3)
  - x < 0 and (y/x) == 3
  - x > 0 or (y/x) == 3

ANSWER – Boolean expressions

- Assume the variable x currently stores the value 47. What will the following expressions evaluate to?
  - True and not False - True
  - not True or not False - True
  - True and not 0 – True
  - False or not 1 - False
  - 2 < x and x < 50 - True
  - x > 2 and < 20 – Invalid syntax
  - (2 > x) or (x == 47) - True
  - not (x == x) - False
  - not (x != 3) – False
  - x < 0 and (y/x) == 3 – False
  - x > 0 or (y/x) == 3 - True

PRACTICE TIME – Conditionals

- Rewrite the following code to make it cleaner and avoid the use of nested if statements:

```
def get_discount(member, purchase_amount):
        if member == True:
                if purchase_amount > 100:
                        return "Discount applied"
                else:
                        return "No discount"
        else:
                if purchase_amount > 200:
                        return "Discount applied"
                else:
                        return "No discount"
```

print(get_discount(True, 120)) # Expected: "Discount applied" print(get_discount(True, 80)) # Expected: "No discount" print(get_discount(False, 220)) # Expected: "Discount applied" print(get_discount(False, 50)) # Expected: "No discount"

ANSWER – Conditionals

- Rewrite the following code to make it cleaner and avoid the use of nested if statements:

```
def get_discount(member, purchase_amount):
        if (purchase_amount > 200) or (member and purchase_amount > 100):
                return "Discount applied"
        else:
                return "No discount"

print(get_discount(True, 120)) # Expected: "Discount applied" print(get_discount(True, 80)) # Expected: "No discount" print(get_discount(False, 220)) # Expected: "Discount applied" print(get_discount(False, 50)) # Expected: "No discount"
```

Practice Problems – Problem 1

- You are given the following truth table for a Boolean function with three variables, A, B, and C, and an output column F.
- What are the disjunctive and conjunctive normal forms based on this truth table?

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Practice Problems – Problem 2

Assuming x = 6, y = 2, z = 12. What would the following Python expressions evaluate to?

- x == 10 or x < 5
- x != 7 and x > 3
- not (x >= 5 and y <= 2)
- (x < 0) or (x >= 0)
- not (x != x)
- x > 1 and (z / x) != 2
- x <= 0 or (z / x) == 1
- x == 4 and y == 4

Practice Problems – Answer 1

- Disjunctive normal form:

  $F = (\neg A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg C) \vee (A \wedge \neg B \wedge C) \vee (A \wedge B \wedge \neg C)$

- Conjunctive normal form:

  $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C) \wedge (\neg A \vee B \vee C) \wedge (\neg A \vee \neg B \vee \neg C)$

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Practice Problems – Answer 2

Assuming x = 6, y = 2, z = 12. What would the following Python expressions evaluate to?

- x == 10 or x < 5 → 6 == 10 or 6 < 5 → False or False → **False**
- x != 7 and x > 3 → 6 != 7 and 6 > 3 → True and True → **True**
- not (x >= 5 and y <= 2) → not (6 >= 5 and 2 <= 2) → not (True and True) → not True → **False**
- (x < 0) or (x >= 0) → 6 < 0 or 6 >= 0 → False or True → **True** (always true, no matter what x is)
- not (x != x) → not False → **True**
- x > 1 and (z / x) != 2 → 6 > 1 and (12 / 6 != 2) → True and False → **False**
- x <= 0 or (z / x) == 1 → 6 <= 0 or (12 / 6 == 1) → False or False → **False**
- x == 4 and y == 4 → 6 == 4 and 2 == 4 → False and False → **False**