PER SCHOLAS

# 300.2.2 - 300.3.1

# Logic, Number Systems and Boolean Operation

# Lesson 1:

## Logic, Number Systems and Binary number, Decimal number, and Hexadecimal

### Learning Objectives:

In this lesson, we will explore and demonstrate number systems.

By the end of this lesson, learners will be able to:

- Describe the Logic and Turing machine.
- Explain bit, byte, and binary numbers.
- Identify a decimal number.
- Identify a hexadecimal number.
- Describe the ASCII Character Set and Unicode.

# Table of content

# Introduction to Logic



- Logic is a science that sets the rules for properly-ordered thinking:
  - Helps with identifying faulty reasoning (fallacies).
  - Helps with verifying a claim validity.
    - The only true way to verify is to test!
- Critical Thinking is the application of those rules:
  - Based on criteria that has been tested and verified.
  - Logically combines criteria to form an argument.

# Why Talk About Logic?



- Logic can be represented in a number of ways
  - Words
  - Math
  - Code or Program
- Computers fundamentally work through **logic.**

How?

# The Turing Machine



- One of the simplest ways of representing logic is through a simple switch.
  - ***On*** or ***Off***, ***Yes*** or ***No***, ***True*** or ***False***, and ***1*** or ***0***.

- A Turing machine consists of a large number of these switches. For instance, let's think of a light that can have four states:
  - **Off**
  - **Red Light**
  - **Yellow Light**
  - **Green Light**

- How many switches would we need to use to produce these states?

# The Turing Machine Switches



**Assume:** We only need two switches. How can we do that?

- Let's call our switches **Left** and **Right**. Each can be **Up** or **Down**.
- So, here is an example of how we could match each state:
  - **Off** – Left Down, Right Down
  - **Red Light** – Left Down, Right Up
  - **Blue Light** – Left Up, Right Down
  - **Yellow Light** – Left Up, Right Up

# A Bit Less Wordy



Okay, great. We were able to set up the light! That being said, our solution was a bit…*wordy*. So, rather than saying Left or Right, let's just say Up or Down for each:

- For Example:
  - **Off** – Down, Down
  - **Red Light** – Down, Up
  - Blue Light – Up, Down
  - **Yellow Light** – Up, Up

# Can we go Further?



Alright, looking good! Let's see if we make that *even shorter*. Let's use **1 for Up** and **0 for Down**, and get rid of the comma.

- For Example:
  - **Off** – 00
  - **Red** – 01
  - **Blue** – 10
  - **Yellow** – 11

# Even More Complex



Congratulations! What you just saw is Binary. What if we wanted to combine colors? For instance, combine Red and Yellow to get Orange.

- In total, we have these states:
  - **Off**
  - **Red**
  - **Purple**
  - **Blue**
  - **Green**
  - **Yellow**
  - **Orange**
  - **Black**
- How many switches would that take?

# More Switches

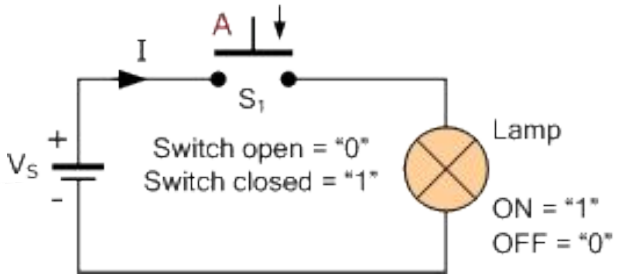


- One answer would be 3.
- In other words, each switch represents a color.
- If a switch is off, that color is not being shown.
- For Example:
  - **Off - 000**
  - **Red - 001**
  - **Purple - 011**
  - **Blue - 010**
  - **Green - 110**
  - **Yellow - 100**
  - **Orange -101**
  - **Black -111**

# Overview of Digital Logic Circuit or Switch



- Digital Logic Circuit is a basic electronic component of a digital system.
- Values of digital signals are **0** or **1** (bits).
- Black Box is specified by the signal input/output table.




| A (Switch) | L (Lamp) |
|---|---|
| 0 | 0 |
| 1 | 1 |
| L = A | |

Click here for more information and details about **Switching Theory of a Switch**

# Bits, Bytes and Binary Number - $(N)_2$

- In Computer Science, a Switch **(0 or 1)** is called a Bit.
- A **binary number** is a number expressed in the ***base-2*** numeral system or binary numeral system. This system uses only two symbols: typically **1** and **0** .
- So, our previous color scheme was represented by 3 Bits.
- A Byte, on the other hand, is represented by **8** bits.
  - For a total of $2^8$ or **256** possible combinations
  - Byte is the smallest addressable unit of memory in most computer architectures.
  - Byte is the smallest variable size in high-level languages like C#, Java and Python.

# Represent number system



The number system is one of the ways to represent numbers

- Binary Numbers – Base 2
- Octal Numbers – Base 8
- Decimal Numbers – Base 10
- Hexadecimal Numbers – Base 16

# Representing Binary to Decimal number

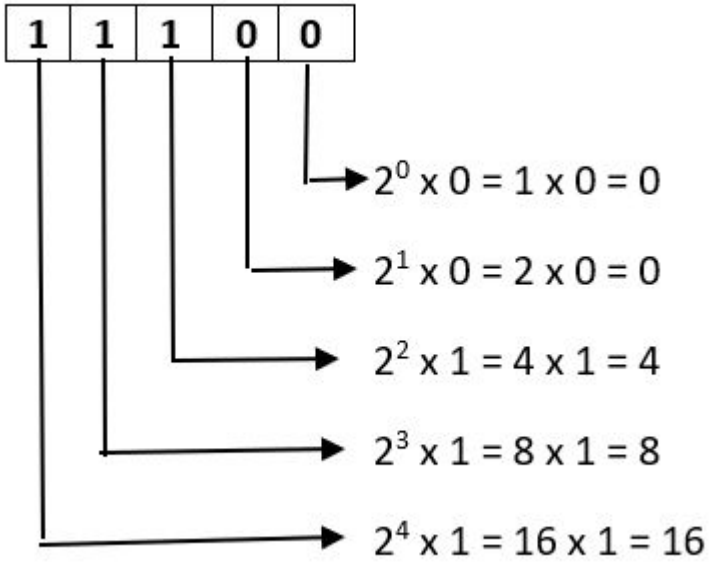


**Example 1:** Convert $(11100)_2 \rightarrow (?)_{10}$

**Solution:**

Resultant Decimal number = **0+0+4+8+16 = 28**

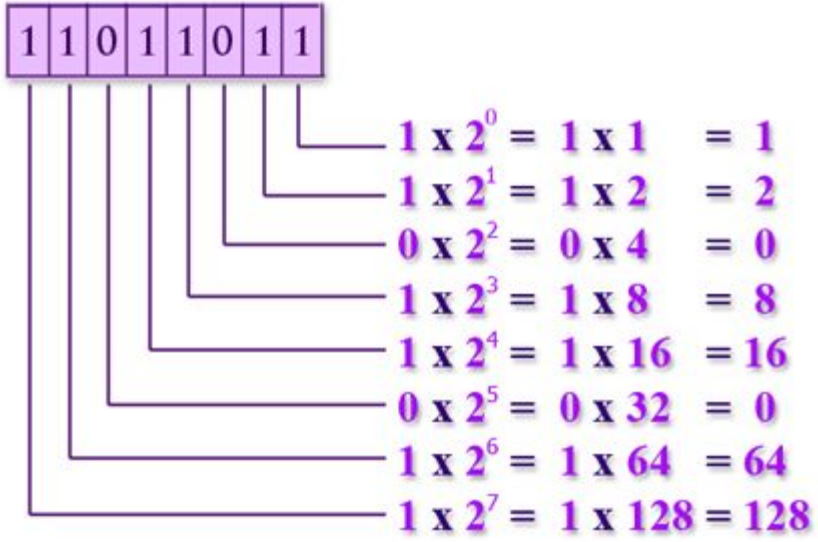
***So, $(11100)_2$->$(28)_{10}$***



# Representing Binary to Decimal number



**Example 2:** Convert a binary number to decimal number.

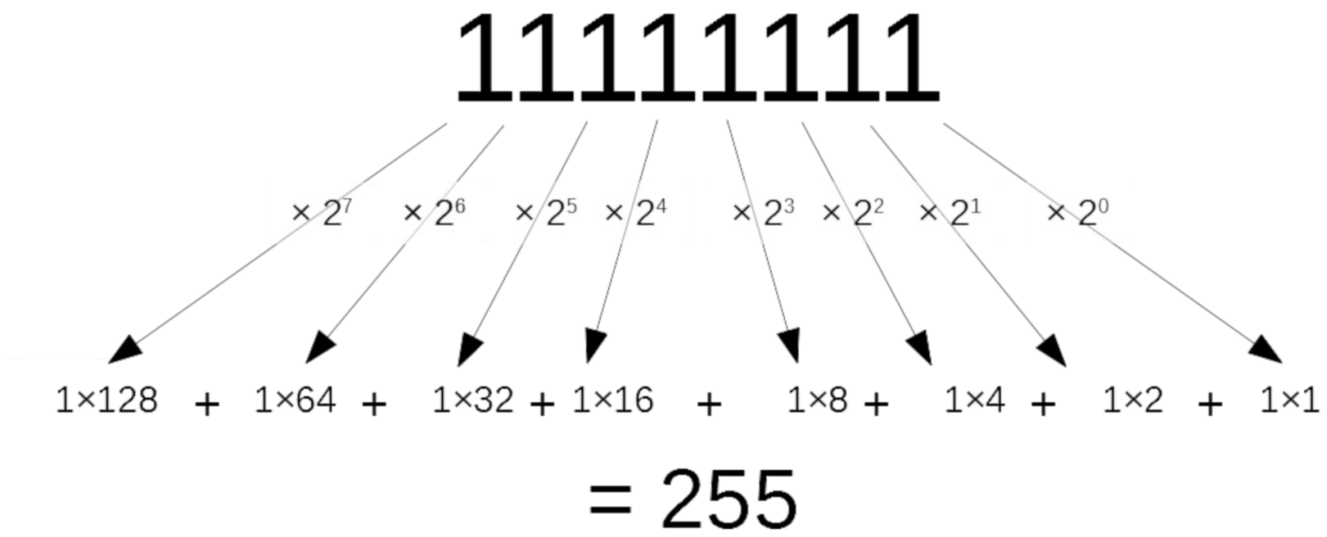



$$1 + 2 + 8 + 16 + 64 + 128 = 219$$

$$(11011011)_2 = (219)_{10}$$

# Representing Binary to Decimal number

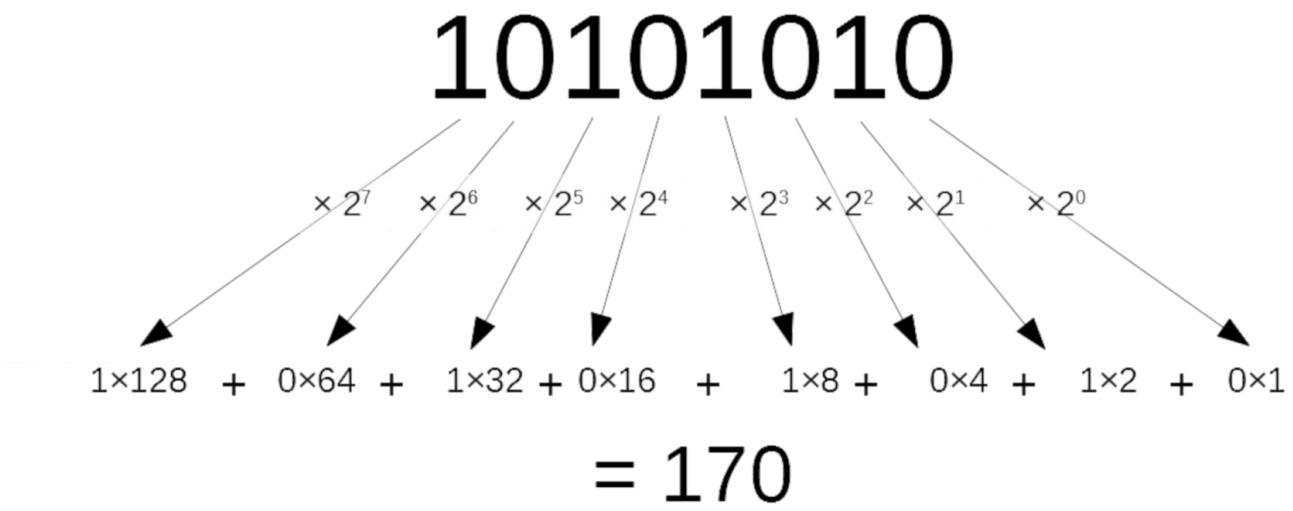
**Example 3:** Convert a binary number to decimal number.




The maximum value we can represent with byte is 255. The minimum is 0. That is a total of 256 values.

# Representing Binary to Decimal number

**Example 4:** Convert a binary number to decimal number.




Every decimal value from 0 to 255 can be represent with a byte.

# Practice Assignment



**Convert following binary numbers to decimal:**

1. $(10)_2 \rightarrow (?)_{10}$
2. $(11)_2 \rightarrow (?)_{10}$
3. $(100)_2 \rightarrow (?)_{10}$
4. $(101)_2 \rightarrow (?)_{10}$
5. $(1000)_2 \rightarrow (?)_{10}$
6. $(1011)_2 \rightarrow (?)_{10}$
7. $(1100)_2 \rightarrow (?)_{10}$
8. $(10101)_2 \rightarrow (?)_{10}$
9. $(11111)_2 \rightarrow (?)_{10}$
10. $(101010)_2 \rightarrow (?)_{10}$
11. $(11100)_2 \rightarrow (?)_{10}$

# Game: Binary Numbers



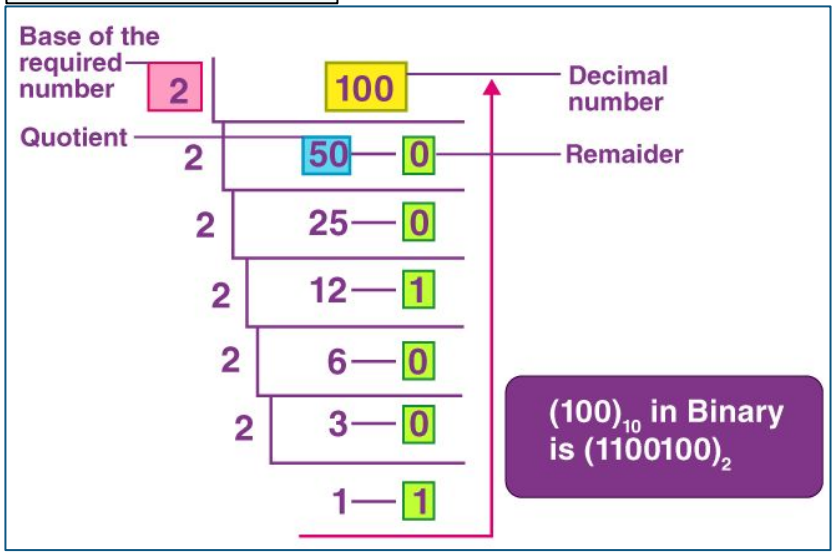
Click here for practice : http://games.penjee.com/binary-numbers-game/

# What is Decimal numbers - $(N)_{10}$ ?

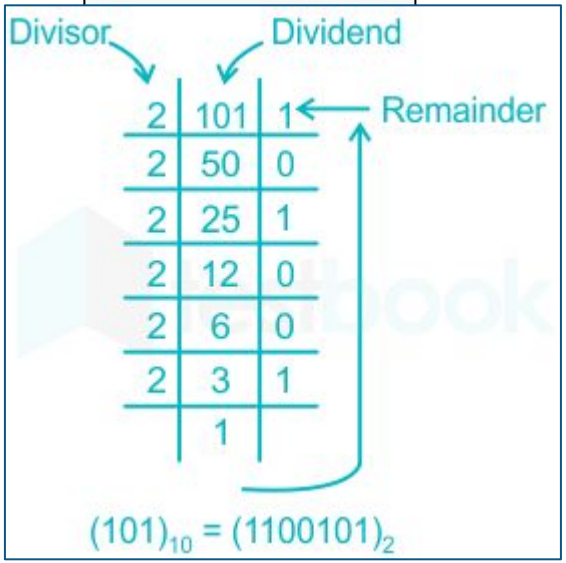


All the **decimal numbers** have their corresponding **binary numbers**. These binary numbers are used in computer applications and used for **programming** or **coding** purposes. This is because binary digits, 0 and 1 are only understood by computers.

**Example one**




**Example Two**




Source; testbook.com/

# What is Standard ASCII?



- ASCII stands for **American Standard Code for Information Interchange**.
- Binary numbers can represent letters and characters. Standard ASCII charts can provide the decoded values without the need to manually convert them. It is important to understand that binary is one of the core encoding types, and knowing how to convert manually can help with validation.

# ASCII character set for binary table



- ASCII tables can reflect all **256 codes** (0–255).
- There will be **95 printable codes**.
  - 52 are US English codes with 26 lowercase and 26 uppercase.
  - The remaining codes represent codes from the decimal 128 decimal values. This is commonly referred to the extended table. The most common table is the ISO 8859-1 which is also called the ISO Latin-1.

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0011 0000 | O | 0100 1111 | m | 0110 1101 |
| 1 | 0011 0001 | P | 0101 0000 | n | 0110 1110 |
| 2 | 0011 0010 | Q | 0101 0001 | o | 0110 1111 |
| 3 | 0011 0011 | R | 0101 0010 | p | 0111 0000 |
| 4 | 0011 0100 | S | 0101 0011 | q | 0111 0001 |
| 5 | 0011 0101 | T | 0101 0100 | r | 0111 0010 |
| 6 | 0011 0110 | U | 0101 0101 | s | 0111 0011 |
| 7 | 0011 0111 | V | 0101 0110 | t | 0111 0100 |
| 8 | 0011 1000 | W | 0101 0111 | u | 0111 0101 |
| 9 | 0011 1001 | X | 0101 1000 | v | 0111 0110 |
| A | 0100 0001 | Y | 0101 1001 | w | 0111 0111 |
| B | 0100 0010 | Z | 0101 1010 | x | 0111 1000 |
| C | 0100 0011 | a | 0110 0001 | y | 0111 1001 |
| D | 0100 0100 | b | 0110 0010 | z | 0111 1010 |
| E | 0100 0101 | c | 0110 0011 | . | 0010 1110 |
| F | 0100 0110 | d | 0110 0100 | , | 0010 0111 |
| G | 0100 0111 | e | 0110 0101 | : | 0011 1010 |
| H | 0100 1000 | f | 0110 0110 | ; | 0011 1011 |
| I | 0100 1001 | g | 0110 0111 | ? | 0011 1111 |
| J | 0100 1010 | h | 0110 1000 | ! | 0010 0001 |
| K | 0100 1011 | I | 0110 1001 | ' | 0010 1100 |
| L | 0100 1100 | j | 0110 1010 | " | 0010 0010 |
| M | 0100 1101 | k | 0110 1011 | ( | 0010 1000 |
| N | 0100 1110 | l | 0110 1100 | ) | 0010 1001 |
| | | | | space | 0010 0000 |

# The ASCII Character Set

| ASCII control characters | | |
|---|---|---|
| 00 | NULL | (Null character) |
| 01 | SOH | (Start of Header) |
| 02 | STX | (Start of Text) |
| 03 | ETX | (End of Text) |
| 04 | EOT | (End of Trans.) |
| 05 | ENQ | (Enquiry) |
| 06 | ACK | (Acknowledgement) |
| 07 | BEL | (Bell) |
| 08 | BS | (Backspace) |
| 09 | HT | (Horizontal Tab) |
| 10 | LF | (Line feed) |
| 11 | VT | (Vertical Tab) |
| 12 | FF | (Form feed) |
| 13 | CR | (Carriage return) |
| 14 | SO | (Shift Out) |
| 15 | SI | (Shift In) |
| 16 | DLE | (Data link escape) |
| 17 | DC1 | (Device control 1) |
| 18 | DC2 | (Device control 2) |
| 19 | DC3 | (Device control 3) |
| 20 | DC4 | (Device control 4) |
| 21 | NAK | (Negative acknowl.) |
| 22 | SYN | (Synchronous idle) |
| 23 | ETB | (End of trans. block) |
| 24 | CAN | (Cancel) |
| 25 | EM | (End of medium) |
| 26 | SUB | (Substitute) |
| 27 | ESC | (Escape) |
| 28 | FS | (File separator) |
| 29 | GS | (Group separator) |
| 30 | RS | (Record separator) |
| 31 | US | (Unit separator) |
| 127 | DEL | (Delete) |

| ASCII printable characters | | | | | |
|---|---|---|---|---|---|
| 32 | space | 64 | @ | 96 | ` |
| 33 | ! | 65 | A | 97 | a |
| 34 | " | 66 | B | 98 | b |
| 35 | # | 67 | C | 99 | c |
| 36 | $ | 68 | D | 100 | d |
| 37 | % | 69 | E | 101 | e |
| 38 | & | 70 | F | 102 | f |
| 39 | ' | 71 | G | 103 | g |
| 40 | ( | 72 | H | 104 | h |
| 41 | ) | 73 | I | 105 | i |
| 42 | * | 74 | J | 106 | j |
| 43 | + | 75 | K | 107 | k |
| 44 | , | 76 | L | 108 | l |
| 45 | - | 77 | M | 109 | m |
| 46 | . | 78 | N | 110 | n |
| 47 | / | 79 | O | 111 | o |
| 48 | 0 | 80 | P | 112 | p |
| 49 | 1 | 81 | Q | 113 | q |
| 50 | 2 | 82 | R | 114 | r |
| 51 | 3 | 83 | S | 115 | s |
| 52 | 4 | 84 | T | 116 | t |
| 53 | 5 | 85 | U | 117 | u |
| 54 | 6 | 86 | V | 118 | v |
| 55 | 7 | 87 | W | 119 | w |
| 56 | 8 | 88 | X | 120 | x |
| 57 | 9 | 89 | Y | 121 | y |
| 58 | : | 90 | Z | 122 | z |
| 59 | ; | 91 | [ | 123 | { |
| 60 | < | 92 | \ | 124 | \| |
| 61 | = | 93 | ] | 125 | } |
| 62 | > | 94 | ^ | 126 | ~ |
| 63 | ? | 95 | _ | | |

| Extended ASCII characters | | | | | | | |
|---|---|---|---|---|---|---|---|
| 128 | Ç | 160 | á | 192 | └ | 224 | Ó |
| 129 | ü | 161 | í | 193 | ┴ | 225 | ß |
| 130 | é | 162 | ó | 194 | ┬ | 226 | Ô |
| 131 | â | 163 | ú | 195 | ├ | 227 | Ò |
| 132 | ä | 164 | ñ | 196 | ─ | 228 | õ |
| 133 | à | 165 | Ñ | 197 | ┼ | 229 | Õ |
| 134 | å | 166 | ª | 198 | ã | 230 | µ |
| 135 | ç | 167 | º | 199 | Ã | 231 | þ |
| 136 | ê | 168 | ¿ | 200 | ╚ | 232 | Þ |
| 137 | ë | 169 | ® | 201 | ╔ | 233 | Ú |
| 138 | è | 170 | ¬ | 202 | ╩ | 234 | Û |
| 139 | ï | 171 | ½ | 203 | ╦ | 235 | Ù |
| 140 | î | 172 | ¼ | 204 | ╠ | 236 | ý |
| 141 | ì | 173 | ¡ | 205 | ═ | 237 | Ý |
| 142 | Ä | 174 | « | 206 | ╬ | 238 | ¯ |
| 143 | Å | 175 | » | 207 | ¤ | 239 | ´ |
| 144 | É | 176 | ░ | 208 | ð | 240 | ≡ |
| 145 | æ | 177 | ▒ | 209 | Ð | 241 | ± |
| 146 | Æ | 178 | ▓ | 210 | Ê | 242 | ‗ |
| 147 | ô | 179 | │ | 211 | Ë | 243 | ¾ |
| 148 | ö | 180 | ┤ | 212 | È | 244 | ¶ |
| 149 | ò | 181 | Á | 213 | ı | 245 | § |
| 150 | û | 182 | Â | 214 | Í | 246 | ÷ |
| 151 | ù | 183 | À | 215 | Î | 247 | ¸ |
| 152 | ÿ | 184 | © | 216 | Ï | 248 | ° |
| 153 | Ö | 185 | ╣ | 217 | ┘ | 249 | ¨ |
| 154 | Ü | 186 | ║ | 218 | ┌ | 250 | · |
| 155 | ø | 187 | ╗ | 219 | █ | 251 | ¹ |
| 156 | £ | 188 | ╝ | 220 | ▄ | 252 | ³ |
| 157 | Ø | 189 | ¢ | 221 | ¦ | 253 | ² |
| 158 | × | 190 | ¥ | 222 | Ì | 254 | ■ |
| 159 | ƒ | 191 | ┐ | 223 | ▀ | 255 | nbsp |

ASCII was originally developed for use with teletype systems, then adapted for use with earlier computer systems.

Modern character encodings incorporate ASCII, but use 16 bits (2 bytes) and include many more characters – up to 65,536 ($2^{16}$).

[Click here for official website]

# Representing Characters and Digits

We can see from the previous slide that our common letters and digits are represented by numbers.

For example, the digit 9 maps to the number 57. The letter Z maps to 90, and z maps to 122.

We are all accustomed to working with numbers using decimal, or "base 10" representation. Using base 10, we can represent all numbers using 10 digits, 0-9.

With binary or "base 2", we can represent all numbers using two digits, 0 and 1.

| Character | Value (Decimal) | Value (Binary) |
|---|---|---|
| 9 | 57 | 00111001 |
| Z | 90 | 01011010 |
| z | 122 | 01111010 |

# What is Hexadecimal Numbers - $(N)_{16}$

- Binary numbers allow us to see the byte structure, but they are difficult to read and write. So computer scientists and programmers use **hexadecimal numbers** or **base 16** to represent numbers.
- Hexadecimal uses digits **0-9** plus letters **A-F** to represent **10 – 15**.
- Hexadecimal numbers are written with the prefix **'0x'**.
- The maximum value of a byte **(255)** is **0xFF**.

# Takeaways for Binary and Hex

- At their core, computers operate through many 0/1 switches.
- In order to represent their values more easily, we use hexadecimal.
- Through logic, combinations of these switches can be bound to different capabilities, such as storing numbers, displaying text, etc.
- Higher level programming languages are *compiled* – analyzed and converted into binary - allowing programmers to provide instructions to machines through program logic.

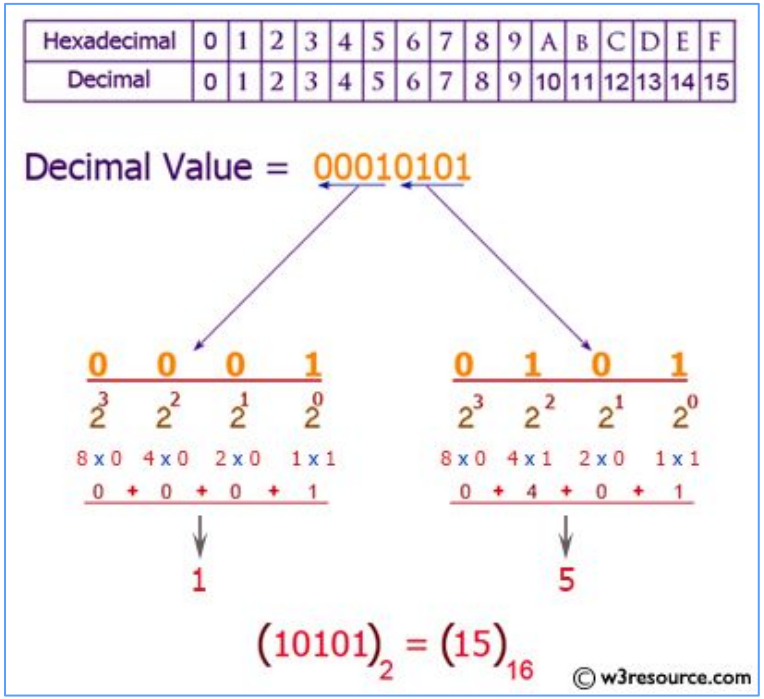
# Example 1 : Converting binary to Hexadecimal



1. Start from the least significant bit **(LSB)** at the right of the binary number and divide it up into groups of **4 digits**(4 digital bits is called a "nibble").
2. Convert each group of 4 binary digits to its equivalent hex value (see in the HEX-DEC table ).
3. Concatenate the results together, giving the total hex number.

**HEX-DEC table**

| Hexadecimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Decimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

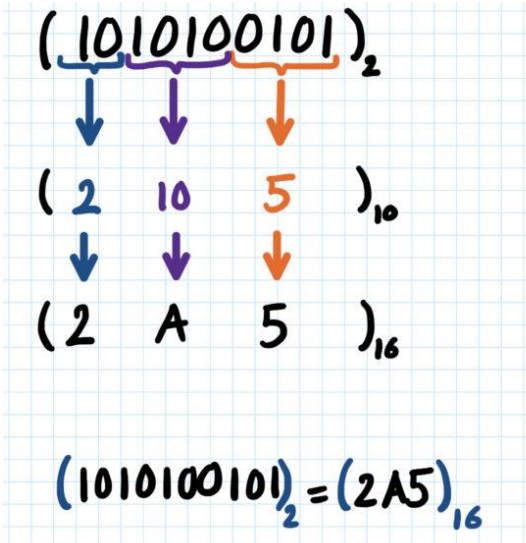


# Example 2 : Converting binary to Hexadecimal



1. Start from the least significant bit **(LSB)** at the right of the binary number and divide it up into groups of **4 digits**(4 digital bits is called a "nibble").
2. Convert each group of 4 binary digits to its equivalent hex value (see in the HEX-DEC table ).
3. Concatenate the results together, giving the total hex number.

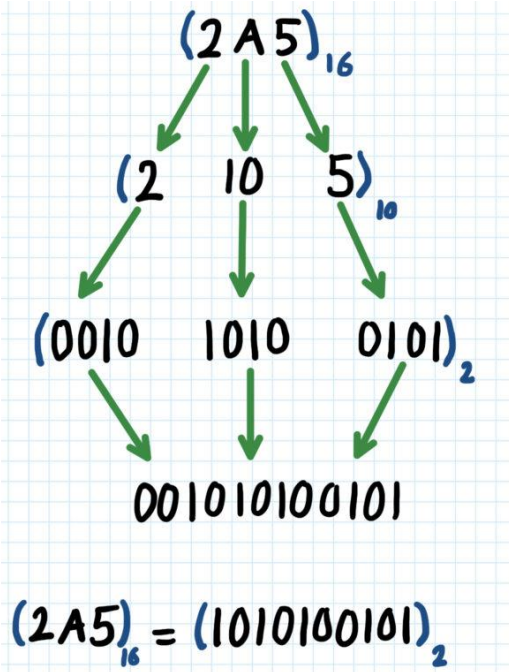
| Hexadecimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Decimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |



# Example 1 : Converting Hexadecimal to binary



| Hexadecimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Decimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |



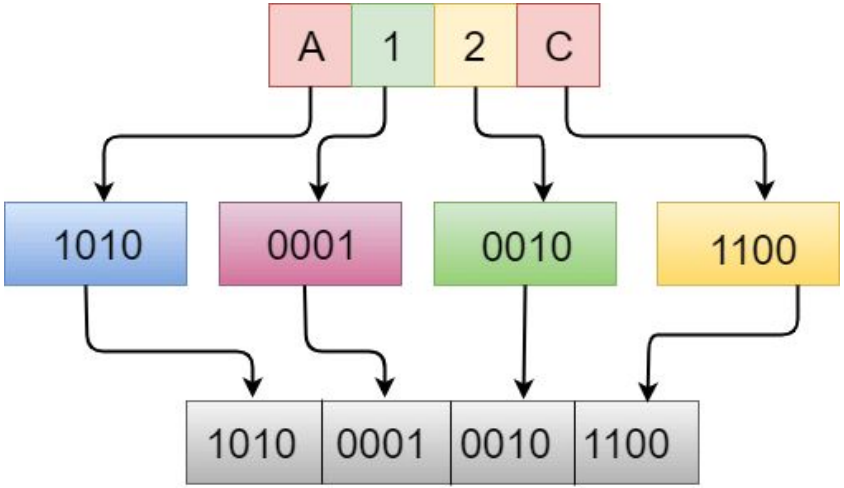
# Example 2 : Converting Hexadecimal to binary



1. Converting Hexadecimal number **$(A12C)_{16}$** into **binary number $(N)_2$**
2. Binary value equivalent to A is 1010
3. Binary value equivalent to 1 is 0001
4. Binary value equivalent to 2 is 0010
5. Binary value equivalent to C is 1100
6. Therefore, the binary value equivalent to **$(A12C)_{16}$** i s **$(1010000100101100)_2$**

**HEX-DEC table**

| Hexadecimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Decimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |



# Representations of the integers from 0 to 255



The following table contains the decimal, 8-bit binary, and 2-digit hex representations of the integers from 0 to 255.

| dec | binary | hex | dec | binary | hex | dec | binary | hex | dec | binary | hex |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00000000 | 00 | 32 | 00100000 | 20 | 64 | 01000000 | 40 | 96 | 01100000 | 60 |
| 1 | 00000001 | 01 | 33 | 00100001 | 21 | 65 | 01000001 | 41 | 97 | 01100001 | 61 |
| 2 | 00000010 | 02 | 34 | 00100010 | 22 | 66 | 01000010 | 42 | 98 | 01100010 | 62 |
| 3 | 00000011 | 03 | 35 | 00100011 | 23 | 67 | 01000011 | 43 | 99 | 01100011 | 63 |
| 4 | 00000100 | 04 | 36 | 00100100 | 24 | 68 | 01000100 | 44 | 100 | 01100100 | 64 |
| 5 | 00000101 | 05 | 37 | 00100101 | 25 | 69 | 01000101 | 45 | 101 | 01100101 | 65 |
| 6 | 00000110 | 06 | 38 | 00100110 | 26 | 70 | 01000110 | 46 | 102 | 01100110 | 66 |
| 7 | 00000111 | 07 | 39 | 00100111 | 27 | 71 | 01000111 | 47 | 103 | 01100111 | 67 |
| 8 | 00001000 | 08 | 40 | 00101000 | 28 | 72 | 01001000 | 48 | 104 | 01101000 | 68 |
| 9 | 00001001 | 09 | 41 | 00101001 | 29 | 73 | 01001001 | 49 | 105 | 01101001 | 69 |
| 10 | 00001010 | 0A | 42 | 00101010 | 2A | 74 | 01001010 | 4A | 106 | 01101010 | 6A |
| 11 | 00001011 | 0B | 43 | 00101011 | 2B | 75 | 01001011 | 4B | 107 | 01101011 | 6B |
| 12 | 00001100 | 0C | 44 | 00101100 | 2C | 76 | 01001100 | 4C | 108 | 01101100 | 6C |
| 13 | 00001101 | 0D | 45 | 00101101 | 2D | 77 | 01001101 | 4D | 109 | 01101101 | 6D |
| 14 | 00001110 | 0E | 46 | 00101110 | 2E | 78 | 01001110 | 4E | 110 | 01101110 | 6E |
| 15 | 00001111 | 0F | 47 | 00101111 | 2F | 79 | 01001111 | 4F | 111 | 01101111 | 6F |
| 16 | 00010000 | 10 | 48 | 00110000 | 30 | 80 | 01010000 | 50 | 112 | 01110000 | 70 |
| 17 | 00010001 | 11 | 49 | 00110001 | 31 | 81 | 01010001 | 51 | 113 | 01110001 | 71 |
| 18 | 00010010 | 12 | 50 | 00110010 | 32 | 82 | 01010010 | 52 | 114 | 01110010 | 72 |
| 19 | 00010011 | 13 | 51 | 00110011 | 33 | 83 | 01010011 | 53 | 115 | 01110011 | 73 |
| 20 | 00010100 | 14 | 52 | 00110100 | 34 | 84 | 01010100 | 54 | 116 | 01110100 | 74 |
| 21 | 00010101 | 15 | 53 | 00110101 | 35 | 85 | 01010101 | 55 | 117 | 01110101 | 75 |
| 22 | 00010110 | 16 | 54 | 00110110 | 36 | 86 | 01010110 | 56 | 118 | 01110110 | 76 |
| 23 | 00010111 | 17 | 55 | 00110111 | 37 | 87 | 01010111 | 57 | 119 | 01110111 | 77 |
| 24 | 00011000 | 18 | 56 | 00111000 | 38 | 88 | 01011000 | 58 | 120 | 01111000 | 78 |
| 25 | 00011001 | 19 | 57 | 00111001 | 39 | 89 | 01011001 | 59 | 121 | 01111001 | 79 |
| 26 | 00011010 | 1A | 58 | 00111010 | 3A | 90 | 01011010 | 5A | 122 | 01111010 | 7A |
| 27 | 00011011 | 1B | 59 | 00111011 | 3B | 91 | 01011011 | 5B | 123 | 01111011 | 7B |
| 28 | 00011100 | 1C | 60 | 00111100 | 3C | 92 | 01011100 | 5C | 124 | 01111100 | 7C |
| 29 | 00011101 | 1D | 61 | 00111101 | 3D | 93 | 01011101 | 5D | 125 | 01111101 | 7D |
| 30 | 00011110 | 1E | 62 | 00111110 | 3E | 94 | 01011110 | 5E | 126 | 01111110 | 7E |
| 31 | 00011111 | 1F | 63 | 00111111 | 3F | 95 | 01011111 | 5F | 127 | 01111111 | 7F |

| dec | binary | hex | dec | binary | hex | dec | binary | hex | dec | binary | hex |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 10000000 | 80 | 160 | 10100000 | A0 | 192 | 11000000 | C0 | 224 | 11100000 | E0 |
| 129 | 10000001 | 81 | 161 | 10100001 | A1 | 193 | 11000001 | C1 | 225 | 11100001 | E1 |
| 130 | 10000010 | 82 | 162 | 10100010 | A2 | 194 | 11000010 | C2 | 226 | 11100010 | E2 |
| 131 | 10000011 | 83 | 163 | 10100011 | A3 | 195 | 11000011 | C3 | 227 | 11100011 | E3 |
| 132 | 10000100 | 84 | 164 | 10100100 | A4 | 196 | 11000100 | C4 | 228 | 11100100 | E4 |
| 133 | 10000101 | 85 | 165 | 10100101 | A5 | 197 | 11000101 | C5 | 229 | 11100101 | E5 |
| 134 | 10000110 | 86 | 166 | 10100110 | A6 | 198 | 11000110 | C6 | 230 | 11100110 | E6 |
| 135 | 10000111 | 87 | 167 | 10100111 | A7 | 199 | 11000111 | C7 | 231 | 11100111 | E7 |
| 136 | 10001000 | 88 | 168 | 10101000 | A8 | 200 | 11001000 | C8 | 232 | 11101000 | E8 |
| 137 | 10001001 | 89 | 169 | 10101001 | A9 | 201 | 11001001 | C9 | 233 | 11101001 | E9 |
| 138 | 10001010 | 8A | 170 | 10101010 | AA | 202 | 11001010 | CA | 234 | 11101010 | EA |
| 139 | 10001011 | 8B | 171 | 10101011 | AB | 203 | 11001011 | CB | 235 | 11101011 | EB |
| 140 | 10001100 | 8C | 172 | 10101100 | AC | 204 | 11001100 | CC | 236 | 11101100 | EC |
| 141 | 10001101 | 8D | 173 | 10101101 | AD | 205 | 11001101 | CD | 237 | 11101101 | ED |
| 142 | 10001110 | 8E | 174 | 10101110 | AE | 206 | 11001110 | CE | 238 | 11101110 | EE |
| 143 | 10001111 | 8F | 175 | 10101111 | AF | 207 | 11001111 | CF | 239 | 11101111 | EF |
| 144 | 10010000 | 90 | 176 | 10110000 | B0 | 208 | 11010000 | D0 | 240 | 11110000 | F0 |
| 145 | 10010001 | 91 | 177 | 10110001 | B1 | 209 | 11010001 | D1 | 241 | 11110001 | F1 |
| 146 | 10010010 | 92 | 178 | 10110010 | B2 | 210 | 11010010 | D2 | 242 | 11110010 | F2 |
| 147 | 10010011 | 93 | 179 | 10110011 | B3 | 211 | 11010011 | D3 | 243 | 11110011 | F3 |
| 148 | 10010100 | 94 | 180 | 10110100 | B4 | 212 | 11010100 | D4 | 244 | 11110100 | F4 |
| 149 | 10010101 | 95 | 181 | 10110101 | B5 | 213 | 11010101 | D5 | 245 | 11110101 | F5 |
| 150 | 10010110 | 96 | 182 | 10110110 | B6 | 214 | 11010110 | D6 | 246 | 11110110 | F6 |
| 151 | 10010111 | 97 | 183 | 10110111 | B7 | 215 | 11010111 | D7 | 247 | 11110111 | F7 |
| 152 | 10011000 | 98 | 184 | 10111000 | B8 | 216 | 11011000 | D8 | 248 | 11111000 | F8 |
| 153 | 10011001 | 99 | 185 | 10111001 | B9 | 217 | 11011001 | D9 | 249 | 11111001 | F9 |
| 154 | 10011010 | 9A | 186 | 10111010 | BA | 218 | 11011010 | DA | 250 | 11111010 | FA |
| 155 | 10011011 | 9B | 187 | 10111011 | BB | 219 | 11011011 | DB | 251 | 11111011 | FB |
| 156 | 10011100 | 9C | 188 | 10111100 | BC | 220 | 11011100 | DC | 252 | 11111100 | FC |
| 157 | 10011101 | 9D | 189 | 10111101 | BD | 221 | 11011101 | DD | 253 | 11111101 | FD |
| 158 | 10011110 | 9E | 190 | 10111110 | BE | 222 | 11011110 | DE | 254 | 11111110 | FE |
| 159 | 10011111 | 9F | 191 | 10111111 | BF | 223 | 11011111 | DF | 255 | 11111111 | FF |

Click here for better view

# Binary to Hexadecimal Converter



Below link is for the convert.
https://www.binaryhexconverter.com/binary-to-hex-converter

# Additional Resources



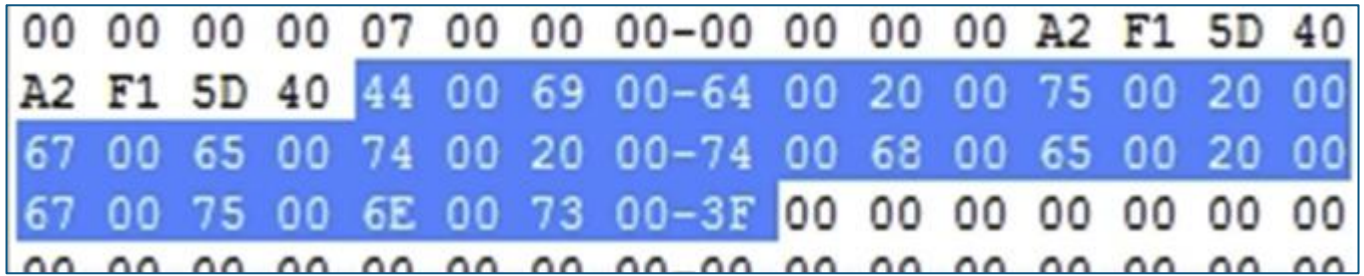
Watch video → Converting Binary to Hexadecimal

Watch video → Converting Hexadecimal to Binary

# Overview of Unicode

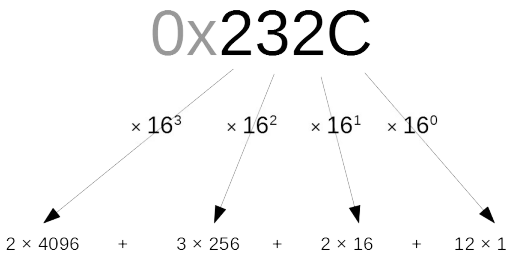


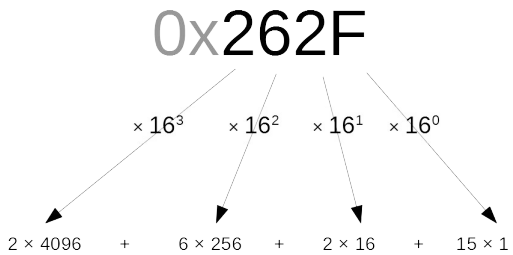
- Unicode is a computing standard for the consistent encoding symbols.
- Languages other than English were limited to the 256 codes of the ASCII. They could not fit into this limited space and developers created Unicode. This creates a 2 bytes number for every character, no matter what language is used.
- Using 16 bits (2 bytes), 65,536 characters can be defined, whereas 256 possibilities were possible with 8 bits (1 byte).
- One way to quickly recognize Unicode is to look for the 00 padding between the bytes. Below screen shot is an example of Unicode

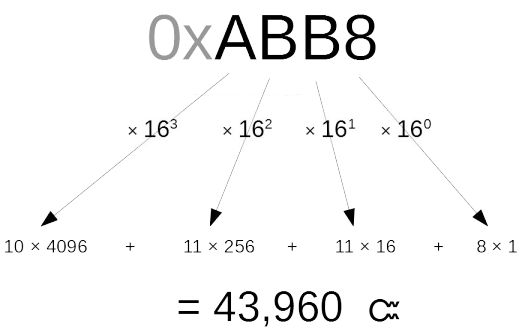


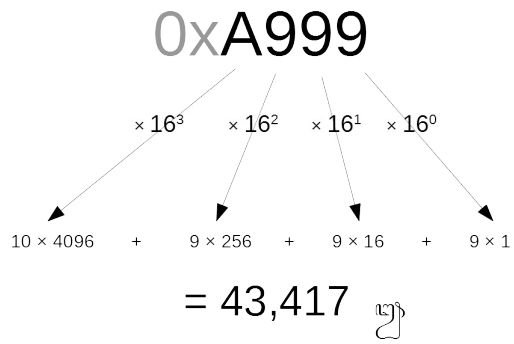

Click here for more explanation and details about **unicode.**

# Hexadecimal Numbers and Unicode



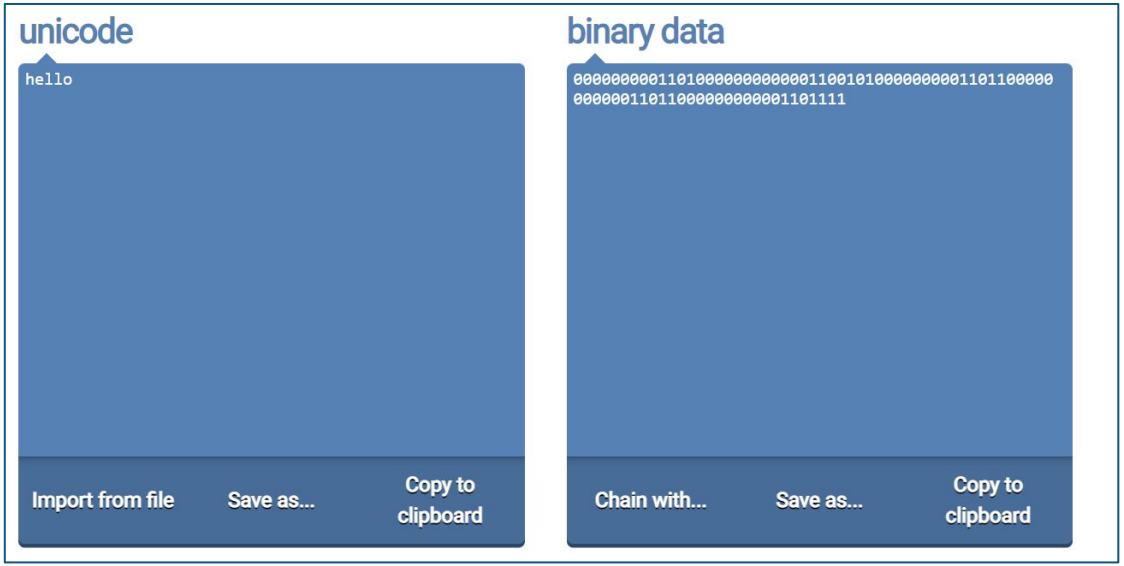

- The Unicode character set uses values from 0x0000 to 0xFFFF – that is 65,536 unique values.
- Using Unicode characters, computers can display information in most of the world's languages and alphabets.
- Unicode also includes many symbols and emojis.
- Click here for unicode Symbols examples

# Unicode Converter



Below link is for the third party tools, Which you can use for demonstration

https://onlineunicodetools.com/convert-unicode-to-binary



# Optional Resources



- Video: How Does A Transistor Work?: https://www.youtube.com/watch?v=IcrBqCFLHIY
- Video: How Does a CPU Work?: https://www.youtube.com/watch?v=cNN_tTXABUA
- Video: How Do Circuit Boards Work?: https://www.youtube.com/watch?v=gUzyd4klMQk

These are additional resources. You can watch these videos in office hours.

# Lesson 2:

## Formal Logic, Boolean operator and Truth Table

### Learning Objective:

In this lesson, we will explore the formal logic and Boolean operators using truth table.

By the end of this presentation, learners will be able to

- Describe and explain formal logic, statement and arguments.
- Understand and describe the boolean logic and operators using truth table.

# Table of contents

# What is formal logic?



- The study of the the form of **arguments**, unobscured by contingent (dependent upon a particular state of events to be true) content.
  - Logic investigates the level of correctness of the reasoning found in arguments.

# What is Statements?



- A **statement** is a sentence that is either **true** or **false**, such as ***"The cat is on the mat."*** Many sentences are not statements, such as ***"Close the door, please"*** , ***"How old are you?"***
- Statements are one example of a premise
- They **MUST** be declarative
  - It must express a truth or possible truth
- For instance:
  - **"What time is it?"** – **NOT a statement**
  - **"Close the door!"** – **NOT a statement**
  - **"All humans are homo sapiens"** – **IS a statement**
  - **"Coffee usually tastes bitter"** – **IS a statement**
- Statements can be combined with **logical operators**
  - Ann is home **OR** Bob is home
  - I made cappuccino this morning **AND** I got to work on time

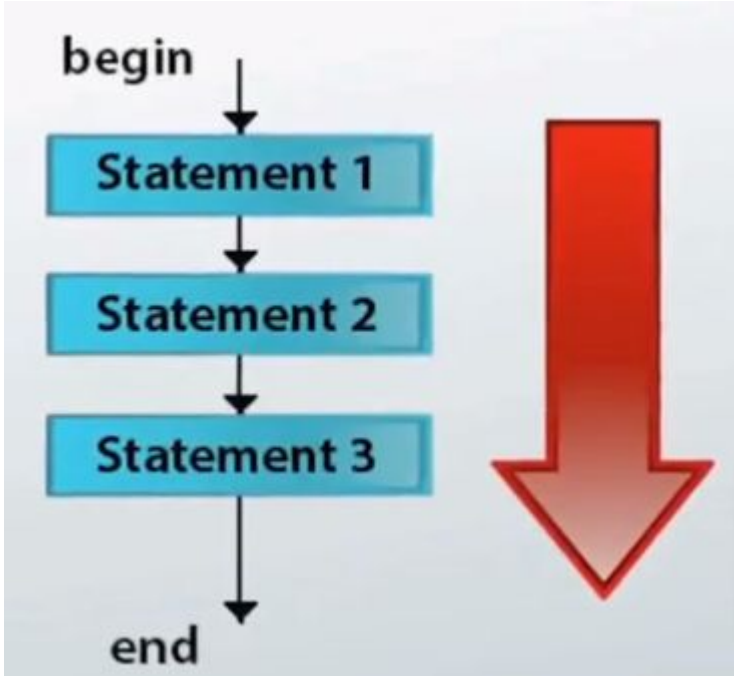
# Statement in Programming



- In programming, Statements are the basic unit of code. A statement can assign a value to a variable, perform a single action, control the execution of other statements and do any number of other things. A **statement** consists of a single line of code that does something. A very simple example of a statement is the following:

**Statement Example:**

```
print("Hello World!");
```



# What is Arguments?



- An **Argument** is a group of statements including one or more **premises** and one and only one **conclusion**. The point of an **argument** is to give the receiver of the argument good reason to believe new information.
  - Arguments are the focal point of logic
  - They are the way we structure our thinking
  - They have a formal structure
  - They consist of reasons in support of a claim

# Structure of an Arguments



- **Premises**
  - A premise that is implied, or is necessary for the argument to be valid, but is unstated.
  - The evidence used in the argument
  - There can be one or many premises in a single argument.
- **Conclusion**
  - A **conclusion** is a statement in an argument that indicates of what the arguer is trying to convince the reader/listener.well.
  - The statement being proved by the premises
- **Logical Relation**
  - What connects the premises to the conclusion

Remember that we are always concerned with the relation between the premise(s) and the conclusion.

# Arguments groups



Arguments can be divided into two groups:

- **deductive arguments - *involve necessity***
- **inductive arguments - *involve probability***

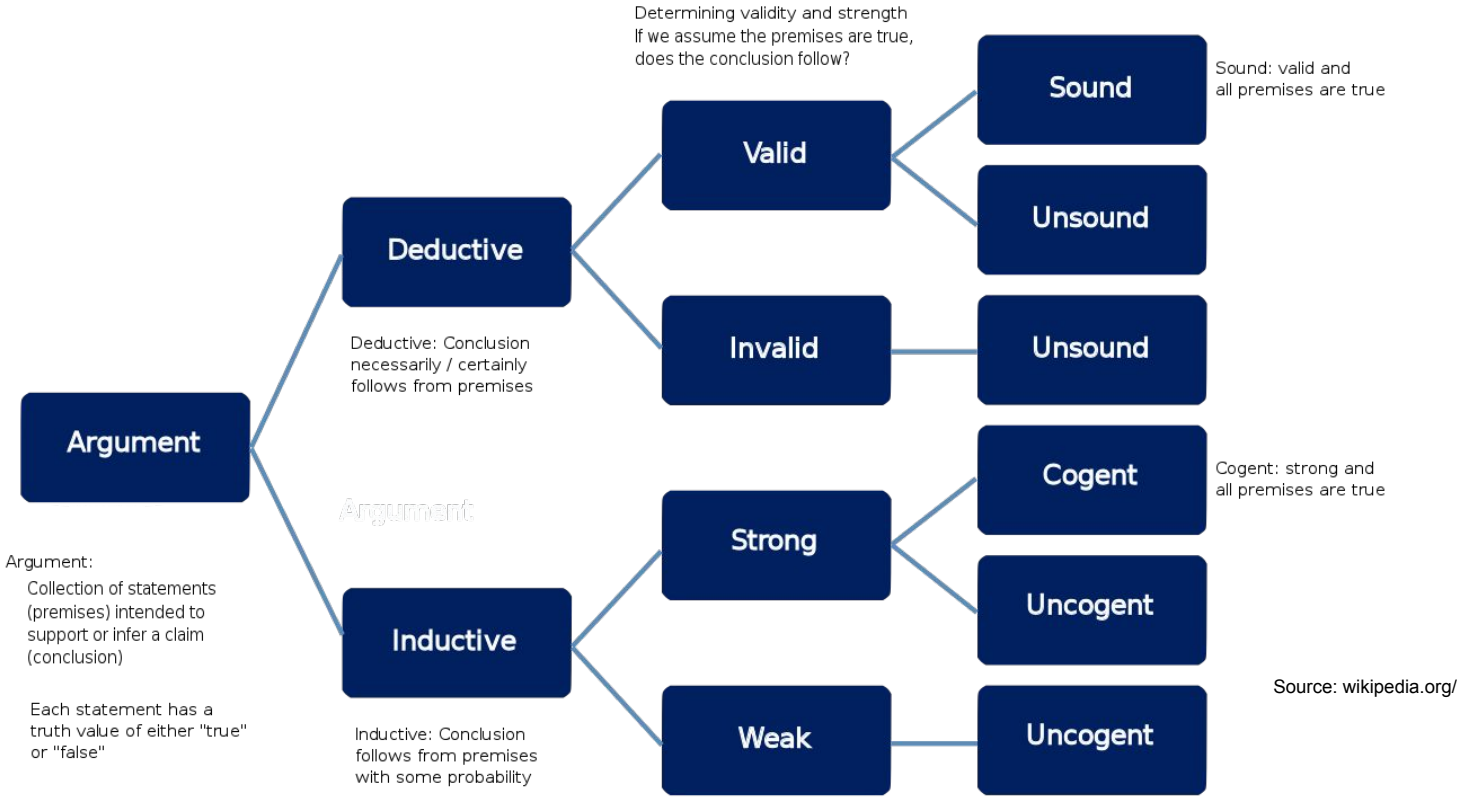
- **Inductive argument:** An argument in which the **premises** are intended to provide probable support for the **conclusion**.

- **Deductive argument:** An argument in which the **premises** are intended to provide necessary support for the **conclusion.** Deductive arguments are either **valid** or **invalid**.

For more information and details about argument groups visit Wiki document

# Arguments groups





# Example of a “right” logical argument



**Example 1**

- Premise 1: All vegetables are plants
- Premise 2: This tomato is a vegetable
- Conclusion: This tomato must be a plant

- Notes:
  - There is a direct relation between the premises
  - The conclusion is a result of that relation

# Example of a "wrong" logical argument



**Example 2**

- Premise 1: Stephen King is an Author
- Premise 2: The sky is blue
- Conclusion: Therefore, it will rain tomorrow morning

- Notes:
  - There is no direct relation between the premises
  - Even if the conclusion was true, it would not have anything to do with the premises

# Example of a "right" logical argument



**Example 3**

## The Argument

**Public libraries should be funded in every community because they provide learning resources for all ages and provide safe spaces for people to read, study, and gather.**

*PREMISE #1*

**they provide learning resources for all ages**

*PREMISE #2*

**provide safe spaces for people to read, study, and gather**

*The CONCLUSION*

**Public libraries should be funded in every community**

# Where "premises" can go wrong



**Example 1**

- Not being specific enough
  - **Invalid logic:**
    - **Premise**: I made cappuccino this morning AND I got to work on time
    - **Conclusion:** Therefore, it's going to be a good day
  - **Valid logic:**
    - **Premise 1:** I made cappuccino this morning AND I got to work on time
    - **Premise 2:** If I make cappuccino in the morning or get to work on time, it's a good day.
    - **Conclusion:** Therefore, it's going to be a good day
- Misunderstanding
  - The epistemic position of the defense side of the litigation lacks justificatory veracity

# Where “premises” can go wrong



**Example 2**

In below statement some details are Missing

- **Premise:** It rained yesterday AND rain causes the roses to bloom.
- **Conclusion:** The roses are blooming today.

- **Problem:** The premise is missing a detail: how long does it take for the rain to cause the roses to bloom?

# What is Propositions?



- A **Proposition** is the meaning **behind the statement.**
- Propositions are sentences or phrases that can be judged to be ***true*** or ***false***, for example ***"The sky is blue" and not "please hurry!"***
- Statements can be reworded while still meaning the same thing
- For instance:
  - The epistemic position of the defense side of the litigation lacks justificatory veracity.
  - The defendant failed to make their case.
- Propositions can remain the same even when changing languages
  - The moon has craters.
  - La luna tiene cráteres.
- The important thing is to make sure people understand your propositions

# Propositions in Formal logic



- A proposition is a **declarative statement**.
- In logic, a proposition can only be **true** or **false**
- Some propositions can be **true** or **false** depending on circumstances, while others are always true or always false
- For instance, compare these:
  - New York City is located in New York State
  - It's raining in Chicago
- We use **T** to denote **TRUE** and **F** to denote **FALSE**.

**Example of propositions:**
John loves vegetables
2+3=5.
2+3=8.
Sun rises from West.

**Example of non-propositions:**
Does John love vegetables?
2 + 3.
Solve the equation 2 + x = 3.
2 + x > 8

# Logical Form: General symbols



- To illustrate the logical form of arguments, we use letters of the alphabet (**p, q,** and **r**) to represent the statements.
- We can use special symbols called **logical connectives, logical operators, propositional operators, or, in classical logic, truth-functional connectives.**
  - A statement is an expression that is **true** or **false** but not both
    - If ***p*** or ***q*** then ***r***

# Example 1 - Logical Form



**Argument :**

"If Jane is a computer science major, then Jane will take SSK3003 course"

**symbolic logic statements**

- ➔ *p* = Jane is a computer science major
- ➔ *q* = Jane will take SSK3003 course
- ➔ The common logical form: If ***p***, then ***q***.

# Example 2 - Logical Form



**Argument :** "If $x < -2$ or $x > 2$, then $x_2 > 4$."

***p*** = $x < -2$,

***q*** = $x > 2$,

***r*** = $x_2 > 4$

**symbolic logic statements**

The common logical form: If **p** or **q**, then **r**.

# Example 3 - Logical Form



**Argument :** "If the program syntax is faulty or if program execution results in division by 0, then the computer will generate an error message. Therefore, if the computer does not generate an error message, then the program syntax is correct and program execution does not result in division by 0."

- *p* = The program syntax faulty.
- *q* = The computer will generate an error message.
- *r* = The program execution results in division by 0.

**symbolic logic statements**

The common logical form:

- If ***p*** and ***q***, then ***r***
- Therefore, if not ***r***, then not ***p*** and not ***q***.

# Overview of Truth Tables



- A truth table is a table whose columns are statements, and whose rows are possible scenarios. The table contains every possible scenario and the truth values that would occur
- A truth table has one column for each input variable (for example, ***p*** and ***q***), and one final column showing all of the possible results of the logical operation that the table represents (for example, **p** AND **q**).
- Each row of the truth table contains one possible configuration of the input variables (for instance, ***p***=true ***q***=false), and the result of the operation for those values.

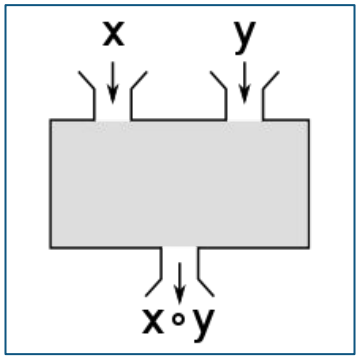
Truth table example

| p | q | p ∧ q | p ∨ q |
|---|---|---|---|
| T | T | T | T |
| T | F | F | T |
| F | T | F | T |
| F | F | F | F |

# Overview of Boolean Logic



- Boolean Logic is a form of algebra which is centered around three simple words known as **Boolean Operators:** **"Or," "And,"** and **"Not"**. At the heart of Boolean Logic is the idea that all values are either **true** or **false.**
  - You start off with the idea that some statement ***P*** is either **true** or ***false***, it can't be anything in between.
  - Then you can form other statements, which are true or false, by combining these initial statements together using the fundamental operators **"Or," "And,"** and **"Not"**.



# Mechanical calculator picture




*An early mechanical calculator. This was one of the first applications of Boolean logic.*

# Binary Operations



There are 16 possible truth functions of two binary variables. Here we will discuss about:

- **Logical Negation operator (¬) ,( ~)** - *both are interchangeable*
- **Logical conjunction (AND)**
- **Logical disjunction (OR)**
- **Logical implication (⟶)**
- **Logical equality (≡)**

# Logical Negation operator (¬),(~)



- Logical negation (¬) is an operation on one logical value, it transforms one proposition to another.
- Negation of ***p*** is ***¬p*** ( alternate ***~p*** )
- Suppose ***p*** is a proposition. The negation of ***p*** is ***¬p***(***~p***)
  - Meaning of ***¬p***: p is **false**

**Truth table of negation - Not P**

| $p$ | $\neg p$ |
|---|---|
| T | F |
| F | T |

**Example:**
John does not love vegetables.
Note that ***¬p*** is a new proposition generated from ***p***.

- We have generated one proposition from another proposition.
- So we call ¬ or ~(the symbol we used to generate the new proposition) the negation operator

# Logical Conjunction operator (^) - AND



- Logical conjunction is an operation on **two logical values**, typically the values of two propositions, that produces a value of **true if both of its operands are true**.
- Now we introduce a binary operator: **conjunction ∧**, which corresponds to **and.**
  - (p **AND** q ) = symbolized by **^**
  - **p ∧ q** is true if and only if **p and q** are both true.
- **Example:**

**Statement:** The sky is blue and the grass is green.

- Conjunction of two primitive statements
- Each single statement gets a letter 1.e. ***p, q***
- Join with **^** i.e. ***p^q***

**Truth table for conjunction**

| $p$ | $q$ | $p \wedge q$ |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

# Logical Disjunction operator (V) - OR



- Logical disjunction is an operation on **two logical values**, typically the values of two propositions, that produces a value of **true if at least one of its operands is true**.

- Another binary operator is disjunction **V**, which corresponds to **OR**, (but is slightly different from common use.)
  - **p V q** is true if and only if **p or q** (or both of them) are true.

- **Example:**
- **Statement:** The sky is blue or the sky is purple

  - → Disjunction of two primitive statements
  - → Each single statement gets a letter i.e. ***p, q***
  - → Join with **v** i.e. ***p v q***

**Truth table for Disjunction**

| $p$ | $q$ | $p \vee q$ |
|---|---|---|
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

# Implication operator (→) - *If, Then*



- **Logical(material conditional)** implication associated with an operation on **two logical values**, typically the values of **two propositions**, which produces a value of **false if the first operand is true** and **the second operand is false**, and **a value of true otherwise**.
- **if p, then q =** Symbolized by implication operator →
- **p → q** corresponds to **p** implies **q**.

**Example:** If this car costs less than $10000, then John will buy it.

**Truth table for Implication:**

| $p$ | $q$ | $p \rightarrow q$ |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | T |
| F | F | T |

Note that :

- when $p$ is **False**, $p \rightarrow q$ is always True.
- $p \rightarrow q$ are equivalent to $\neg p \vee q$.

# Terminology for implication.



**Example**

Proposition ***p***: Alice is smart.

Proposition ***q***: Alice is honest.

- **p → q**
  - That Alice is smart is sufficient for Alice to be honest.
  - "Alice is honest" if "Alice is smart".
- **q → p**
  - That Alice is smart is necessary for Alice to be honest.
  - "Alice is honest" only if "Alice is smart".

# **Bidirectional** implication operator (↔)



- A logical binary operation that returns **true if both the inputs are same(either both true or both false)** and returns **false if one of the input is true** while the other is false.
- Another binary operator bidirectional implication ↔
  - **p** ↔ **q** corresponds to ***p*** is **T** if and only if ***q*** is **T.**

**Example:** A student gets A in bachelor if and only if his grand total is ≥ 95%

**Truth table for for bidirectional implication**

| $p$ | $q$ | $p \leftrightarrow q$ |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | T |

# Summary of Binary operators



We will discovered six different types of sentences in propositional logic for binary, Let's learn how to translate from English to symbols and vice versa with ease.

| Sentence Type | Symbolic Logic |
|---|---|
| Simple | $p$ |
| Negation | $\sim p$ |
| Conjunction | $p \wedge q$ |
| Disjunction | $p \vee q$ |
| Conditional | $p \rightarrow q$ |
| Biconditional | $p \leftrightarrow q$ |

# Formal Languages - Examples



For example, the meaning of the statements **it is raining** and **I am indoors** is transformed when the two are combined with logical connectives.
For statement **P** = **It is raining** and **Q** = **I am indoors**

- **It is not raining (P)**
- **It is raining and I am indoors (P ∧ Q)**
- **It is raining or I am indoors (P ∨ Q)**
- **If it is raining, then I am indoors (P → Q)**
- **If I am indoors, then it is raining (Q → P)**
- **I am indoors if and only if it is raining (P ↔ Q)**

# Truth table - logical form



- We summarized all the possible truth values of a statement in truth tables.
- Truth tables for operators can be
  - Alone
  - Combined
  - Using 0's or 1's

| $p$ | $q$ | $p \wedge q$ | $p \vee q$ | $\sim p$ |
|---|---|---|---|---|
| F | F | F | F | T |
| F | T | F | T | T |
| T | F | F | T | F |
| T | T | T | T | F |

# Precedence of Operators



| | |
|---|---|
| ¬ | 1 |
| ∧ | 2 |
| ∨ | 3 |
| → | 4 |
| ↔ | 5 |

**Example:**

- $\neg p \wedge q$ **means** $(\neg p) \wedge q$
- $p \wedge q \rightarrow r$ **means** $(p \wedge q) \rightarrow r$

**When in doubt, use parenthesis**

- `~ A ^ B` is not the same as `~(A ^ B)`
- `(A ^ B) v C` is not the same as `A^(B v C)` so the parentheses are necessary to avoid ambiguity.

# Translating logical formulas from English sentences



**Example**

- We can also go in the other direction, translating English sentences to logical formulas:
  - Alice is either smart or honest, but Alice is not honest if she is smart:
    - $(p \vee q) \wedge (p \rightarrow \neg q)$
  - That Alice is smart is necessary and sufficient for Alice to be honest:
    - $(p \rightarrow q) \wedge (q \rightarrow p)$.
    - (This is often written as $p \leftrightarrow q$).

# Tautology and Logical equivalence (≡)



**Definitions:**

- A compound proposition that is always True is called a **tautology**.
- Two propositions ***p*** and ***q*** are logically equivalent if their truth tables are the same.
- Namely, ***p*** and ***q*** are logically equivalent if $p \leftrightarrow q$ is a tautology.
- If ***p*** and ***q*** are logically equivalent, we write $p \equiv q$.

$$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$$

Equivalent

# Examples of Logical equivalence



Look at the following two propositions: **$p \rightarrow q$** and **$q \vee \neg p$**

| $p$ | $q$ | $p \rightarrow q$ |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | T |
| F | F | T |

| $p$ | $q$ | $\neg p$ | $q \vee \neg p$ |
|---|---|---|---|
| T | T | F | T |
| T | F | F | F |
| F | T | T | T |
| F | F | T | T |

- The last column of the two truth tables are identical. Therefore **$(p \rightarrow q)$** and **$(q \vee \neg p)$** are logically equivalent.
- So **$(p \rightarrow q) \leftrightarrow (q \vee \neg p)$** is a tautology.
- Thus: **$(p \rightarrow q) \equiv (q \vee \neg p)$**.

# Example of invalid argument form



**Premises:** $p \rightarrow q \vee \sim r$ and $q \rightarrow p \wedge r$, **conclusion:** $p \rightarrow r$

| $p \rightarrow q \vee \sim r$ | $q \rightarrow p \wedge r$ | $p \rightarrow r$ |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | T |
| T | T | F |
| T | F | T |
| T | F | T |
| T | T | T |
| T | T | T |

This row shows it is possible for this argument to have true premises and false conclusion. Hence this form of argument is invalid

# Example of valid argument form



**Premises: p v (q v r)** and **~r**, **conclusion**: **p v r**

| p ∨ (q ∨ r) | ~r | p ∨ r |
|---|---|---|
| T | F | T |
| T | T | T |
| T | F | T |
| T | T | T |
| T | F | T |
| T | T | T |
| T | F | F |
| F | T | F |

# Example for Practice



| $p$ | $q$ | $\neg p$ | $p \wedge q$ | $q \rightarrow \neg p$ | $(p \wedge q) \vee (q \rightarrow \neg p)$ |
|---|---|---|---|---|---|
| T | T | F | T | F | T |
| T | F | F | F | T | T |
| F | T | T | F | T | T |
| F | F | T | F | T | T |

# Practice Exercise - 1



Truth tables can be used to prove many other logical equivalences. For example, consider the following truth table:

| p | q | $p \wedge q$ | $\neg(p \wedge q)$ | $\neg p$ | $\neg q$ | $(\neg p) \vee (\neg q)$ |
|---|---|---|---|---|---|---|
| T | T | | | | | |
| T | F | | | | | |
| F | T | | | | | |
| F | F | | | | | |

# Solution for Practice Exercise - 1



**Solution;**

| p | q | p ∧ q | ¬(p ∧ q) | ¬p | ¬q | (¬p) ∨ (¬q) |
|---|---|---|---|---|---|---|
| T | T | T | F | F | F | F |
| T | F | F | T | F | T | T |
| F | T | F | T | T | F | T |
| F | F | F | T | T | T | T |

You can notice that the negation of a **conjunction: ¬(*p* ∧ *q*)**, and the **disjunction of negations: (¬*p*) ∨ (¬*q*)** are equal

# Practice Exercise - 2



Consider the following truth table:

| p | q | p ∨ q | ¬(p ∨ q) | ¬p | ¬q | (¬p) ∧ (¬q) |
|---|---|---|---|---|---|---|
| T | T | | | | | |
| T | F | | | | | |
| F | T | | | | | |
| F | F | | | | | |

# Solution for Practice Exercise - 2



| p | q | p ∨ q | ¬(p ∨ q) | ¬p | ¬q | (¬p) ∧ (¬q) |
|---|---|---|---|---|---|---|
| T | T | T | F | F | F | F |
| T | F | T | F | F | T | F |
| F | T | T | F | T | F | F |
| F | F | F | T | T | T | T |

# Practice Exercise - 3



Construct a truth table for the :***(p ^ q)  v  ~ r***

| $p$ | $q$ | $r$ | $p \wedge q$ | $\sim r$ | $(p \wedge q) \vee \sim r$ |
|---|---|---|---|---|---|
| F | F | F | F | T | |
| F | F | T | F | F | |
| F | T | F | F | T | |
| F | T | T | F | F | |
| T | F | F | F | T | |
| T | F | T | F | F | |
| T | T | F | T | T | |
| T | T | T | T | F | |

# Solution for Practice Exercise - 3



| $p$ | $q$ | $r$ | $p \wedge q$ | $\sim r$ | $(p \wedge q) \vee \sim r$ |
|---|---|---|---|---|---|
| F | F | F | F | T | T |
| F | F | T | F | F | F |
| F | T | F | F | T | T |
| F | T | T | F | F | F |
| T | F | F | F | T | T |
| T | F | T | F | F | F |
| T | T | F | T | T | T |
| T | T | T | T | F | T |

# Introducing Symbolism



- In mathematics, we can use symbols in equations and formulas
- We can replace those symbols with a variety of numbers and the equations still work
- For example, we could create an addition formula using symbols:
  - `x + 1 = y`

# Example of Symbolism



- Original Argument:
  - All humans are rational
  - All rational things are conscious
  - Therefore, all humans are conscious
- Symbolized Argument:
  - Let **H** stand for human, **R** stand for rational, and **C** stand for conscious and write our argument as:
  - All **H** are **R**
  - All **R** are **C**
  - Therefore All **H** are **C**

# Linking Symbolism to Programming Code



- Original Argument
  - Let **X** stand for an integer and **Y** stand for a second integer
  - The values of **X** and **Y** will always be given as integers
  - Integers can be added together, resulting in another integer
  - Therefore, **X** and **Y** can be added together
  - Return **X+Y**
- Argument in Java

```
int adder(int x, int y){
 return x+y;
}
```

# Summary



- **Logic** is the science that evaluates arguments.
- An **argument** is a group of statements including one or more premises and one and only one conclusion.
- A **statement** is a sentence that is either true or false, such as "The cat is on the mat." Many sentences are not statements, such as "Close the door, please" , "How old are you?"
- A **premise** is a statement in an argument that provides reason or support for the conclusion. There can be one or many premises in a single argument.
- **A conclusion** is a statement in an argument that indicates of what the arguer is trying to convince the reader/listener. What is the argument trying to prove? There can be only one conclusion in a single argument.
- **Logical Form: general rules**
  - All logical comparisons must be done with complete statements
  - A statement is an expression that is true or false but not both
    - If p or q then r
  - If I arrive early or I work hard then I will be promoted
  - **Tautologies and Contradictions**
    - A Tautology (t) is a statement that is always true
    - A Contradiction (c) is a statement that is always false

# Grocery / Synonym



- OS: Operating system
- DB: Database
- HAL: Hardware abstraction layer
- Servers: Servers are defined as fast processing devices
- Workstations: Workstations are also called client computers.
- Networking Devices: Workstations and servers are interconnected with each other by means of a specific medium.
- DBMS: Database Management System

# References



https://en.wikipedia.org/wiki/Client%E2%80%93server_model
https://www.techtarget.com/searchdatamanagement/definition/database
https://www.sciencedirect.com/topics/computer-science/heuristic-algorithm
https://www.baeldung.com/cs/p-np-np-complete-np-hard
https://medium.com/@p.yun1994/p-np-np-hard-and-np-complete-problems-fe679bd1cf9c
https://en.wikipedia.org/wiki/ASCII
https://en.wikipedia.org/wiki/International_Organization_for_Standardization