Data Representation

- How are the values of the different data types stored in the computer ?
  - In memory , in files, in a database, on the network
  - **10101000111010101010010101010101000010101010010**

"Hello World"

"你好世界" "

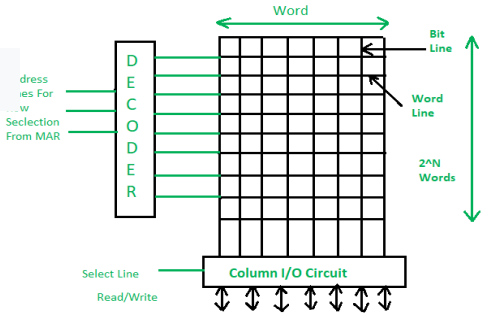
"வணக்கம் உலகம்"

😁 🙏 ➡️ ➡️ 👥 🙌

1234
12.345
True/False

```
{"name":"John",
 "age": 24,
 "hobbies": [
    "chess"
 ]
}
```

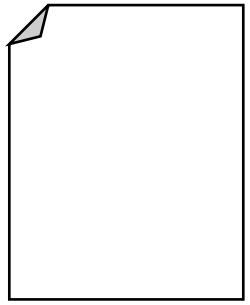
memory



2D Memory Organization

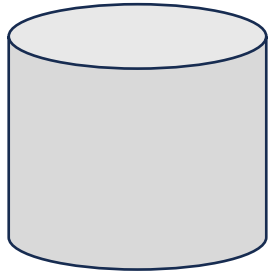
```
message = input()
```

file



```
f.write(message)
```

database



```
con.execute(
f"INSERT
INTO DB
VALUES
{message}")
```

network

```
socket.send(
message.encode()
)
```

Encoding and Decoding

Encoding is the process of converting data from one format to another for storage, transmission and **processing efficiency**.

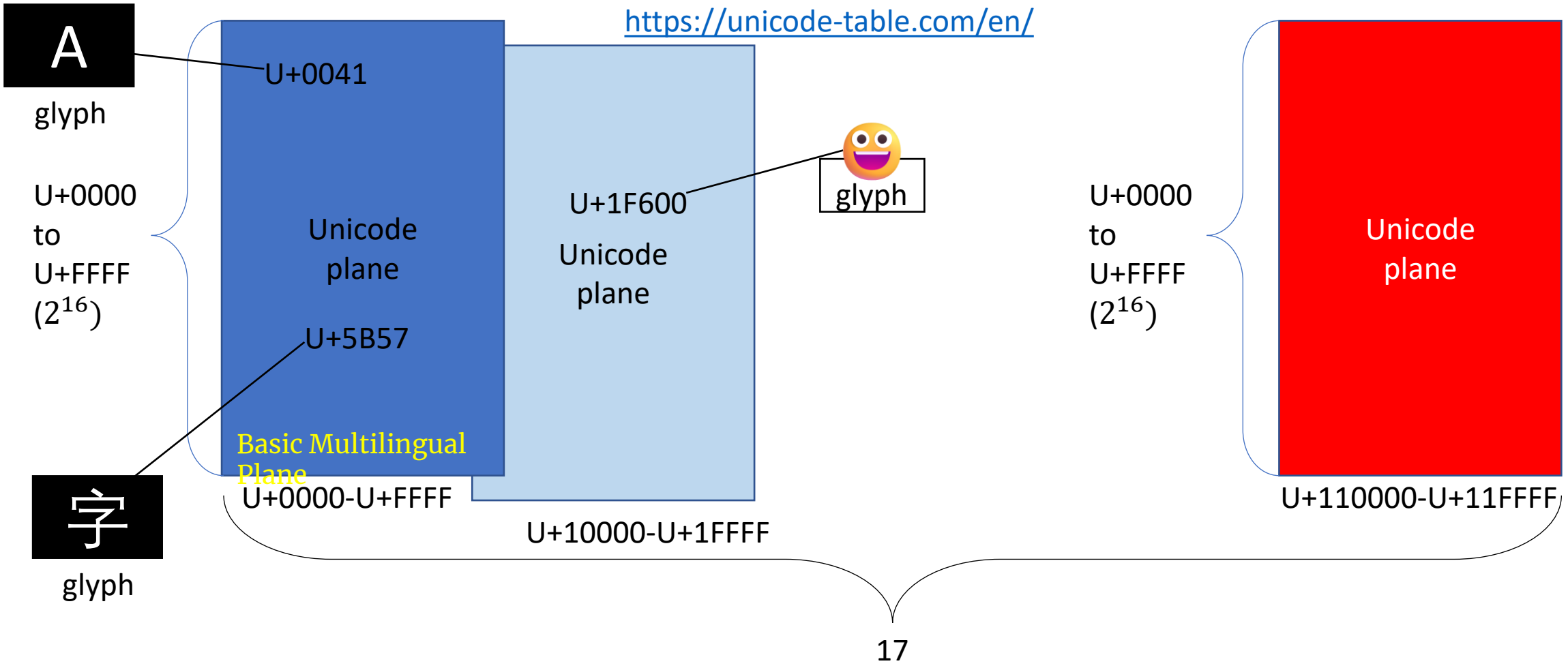
Decoding is the reverse process, converting encoded data back into its original format.

Please **do not confuse** these with the concepts of **Encrypting and Decrypting.** They have different objectives.

Encoding and Decoding is **NEVER** used for the purpose of obfuscation.

Unicode

- every symbol used in human written communication is represented by a non-negative integer known as a code point
- Each code point can be encoded into 3 different encoding scheme:
  - UTF-8 (Default in Python and most operating systems)
  - UTF-16
  - UTF-32



https://unicode-table.com/en/

Total no. of Code Points = $2^{16} \times 17 = 1{,}114{,}112$

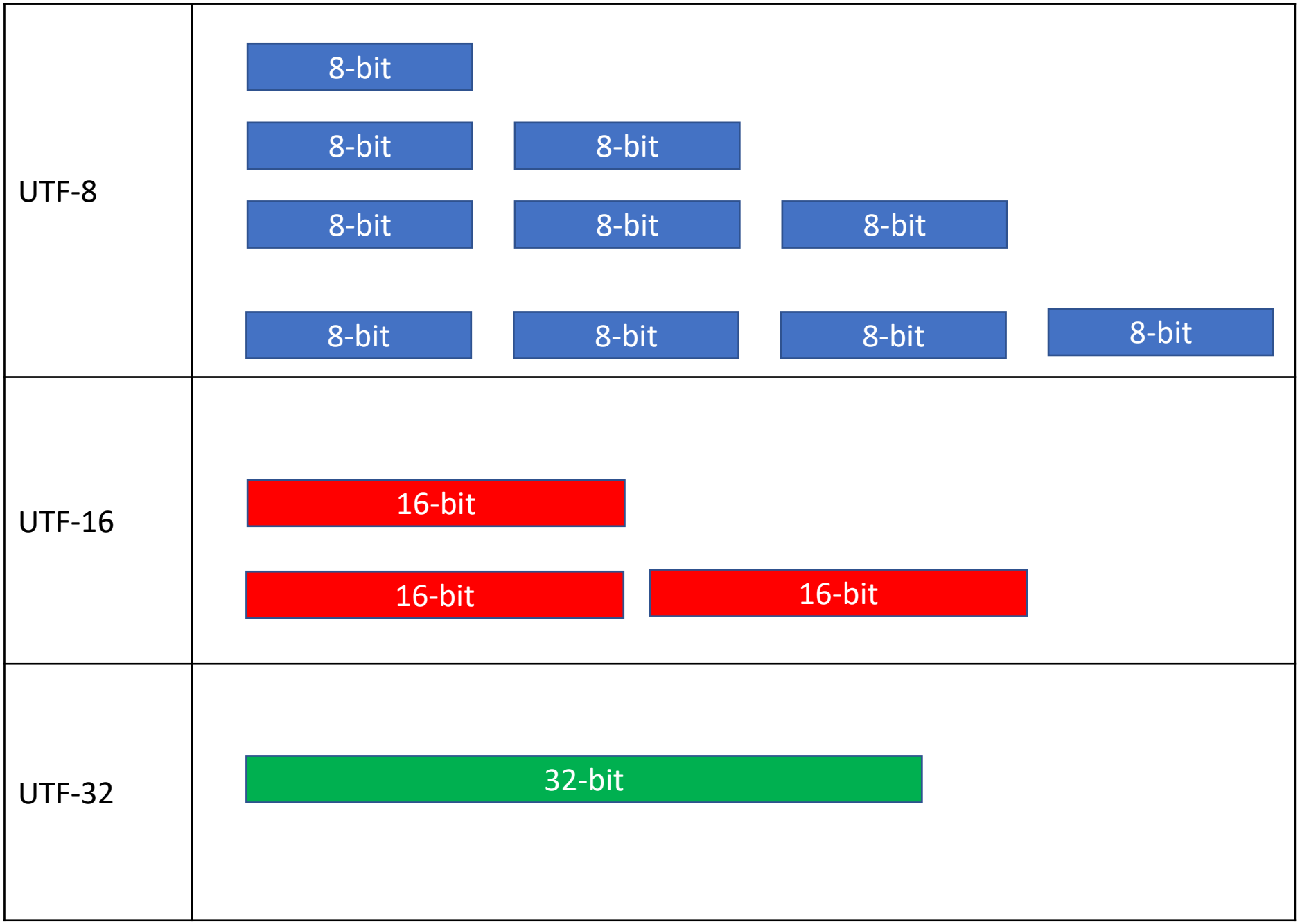
Number of bytes to represent Unicode codepoints = 3 bytes
16 bits + 5 bits(17) = 21 bits

Python Code Points

- Unicode Code Points are written as U+XXXXXX, where X is a hexadecimal digit
- In Python, code points are written as "\uXXXX", if the code point is between 0000 and FFFF and "\UXXXXXXXX" if the code point value is > FFFF
- In Python using string literals
  - `print("\u4e50"), print("\U0001F600")`
- In Python using the chr() function
  - `print( chr(0x4e50) )`

Encoding a Code Point

- storing an integer ranging from 0 to 1 114 112_{10} in memory or file
- 00000000 to 00010000 11111111 11111111_2
- UTF-8, UTF-16, UTF-32 can be used to encode all UNICODE Code points


UTF-8
8-bit
8-bit
8-bit
8-bit
8-bit
8-bit
8-bit
8-bit
8-bit
UTF-16
16-bit
16-bit
16-bit
UTF-32
32-bit

UTF-8 vs UTF-16 vs UTF-32

	Code range	UTF-8	UTF-16	UTF-32
ASCII	U+0000 - U+007F	1	2	4
Cyrillic,Hebrew,Arabic	U+0080 - U+07FF	2		
Thai,CJK Ideographs	U+0800 - U+FFFF	3		
Emoji	U+10000 - U+10FFFF	4	4	

- UTF-8 is compatible with ASCII
- UTF-16 can be used for most languages with just 2 bytes instead of 3 bytes for UTF-8
- UTF-32 seldom used
- BOM, Byte Order Marker is used for UTF-16 and UTF-32 to indicate **Endianness**
  - **0xFF 0xFE ,Little Endian (Least Significant Byte First)**
  - **0xFE 0xFF ,Big Endian (Most Significant Byte First)**