Base64 Encoding & Decoding: Full Details Deep Dive

What is Base64?

Base64 is an encoding scheme that converts binary data into a text format using a set of 64 different ASCII characters. It is commonly used to encode data for transmission over text-based protocols such as email (MIME) and HTTP.

Base64 Character Set

Base64 represents binary data in an ASCII string format using:

- **Uppercase letters**: `A-Z` (26 characters)
- **Lowercase letters**: `a-z` (26 characters)
- **Digits**: `0-9` (10 characters)
- **Symbols**: `+` and `/` (2 characters)
- **Padding Character**: `=` (used when data is not a multiple of 3 bytes)

Total: **64 characters + `=` for padding**

How Base64 Works

1. **Convert Binary Data to ASCII**: Data (text or binary) is broken into 6-bit chunks.
2. **Mapping to Base64 Table**: Each 6-bit chunk is mapped to a character in the Base64 character set.
3. **Padding**: If the original data is not a multiple of 3 bytes, `=` padding is added to make it so.

Base64 Encoding Process

1. Convert input text or binary data to **binary representation**.
2. Split the binary data into **6-bit groups**.
3. Map each 6-bit group to a corresponding **Base64 character**.
4. If the input is **not a multiple of 3 bytes**, add `=` padding.

Example

Encoding the word **"Cat"**:

1. ASCII representation: `C = 67`, `a = 97`, `t = 116`
2. Binary: `67 → 01000011`, `97 → 01100001`, `116 → 01110100`

3. Combine: `010000110110000101110100`
4. Split into 6-bit groups:

```
010000 → 16 (Q)
110110 → 54 (2)
000101 → 5 (F)
110100 → 52 (0)
```

5. Base64 encoded output: "**Q2F0**"

Base64 Decoding Process

1. Reverse the process by mapping Base64 characters **back to 6-bit binary**.
2. Combine the binary groups into **8-bit bytes**.
3. Convert the bytes back to **ASCII characters** or binary data.

Padding Rules

- If **1 byte** remains → Add **2 `=` characters** (E.g., "TQ==").
- If **2 bytes** remain → Add **1 `=` character** (E.g., "TWE=").
- If the input is a **multiple of 3**, no padding is needed.

Applications of Base64

1. **Email Attachments (MIME Encoding)**
2. **Embedding Images in HTML/CSS (``)**
3. **Encoding Binary Data in JSON APIs**
4. **Storing Passwords (NOT recommended for security, use hashing instead)**
5. **Obfuscating Data (But NOT encrypting it!)**

Limitations of Base64

1. **Increases Data Size**: Encoded output is ~33% larger than the original data.
2. **Not Secure**: Base64 is **not encryption**, just encoding.
3. **Processing Overhead**: Extra conversion steps can slow down applications.

Base64 in Python

Encoding

```python
import base64

text = "Hello, Vicky!"
```

```
encoded = base64.b64encode(text.encode())
print(encoded.decode())  # Output: SGVsbG8sIFZpY2t5IQ==
```

Decoding

```
decoded = base64.b64decode(encoded).decode()
print(decoded)  # Output: Hello, Vicky!
```

ENCODING PROCESS (TEXT TO BINARY BYTES)

Base64 Encoding Process

1.Convert input text or binary data to binary representation.

2.Split the binary data into 6-bit groups.

3.Map each 6-bit group to a corresponding Base64 character.

4. If the input is not a multiple of 3 bytes, add = padding.

1 BYTE equal to 8 BITS

```
import base64
# 1 BYTE equal to  8 BITS
text="HI i am vigneshwaran "

encode=base64.b64encode(text.encode())

print(encode)

b'SEkgaSBhbSB2aWduZXNod2FyYW4g'
```

ENCODING PROCESS (DECODE THE BINARY TO STRING)

```
encode=base64.b64encode(text.encode()).decode()
print(f"DECODE THE BINARY TO STRING  :{encode}")
print("sucessfully Binary to string ")

# OR

encode=base64.b64encode(text.encode())
print(encode.decode())
```

```
print(("DONE"))

# import base64
# text="hi i am vicky" #original text
# #utf-8 bites covertion step_1

# utf_code=(text.encode()) #default utf-8
# print(f"step: 1  bites==frist_out>{utf_code}")


# #bites to string step_2
# base64_encode=base64.b64encode(utf_code)
# print(f"step:2  bites to string{base64_encode}")


# #decode process
# base64_decode=base64.b64decode(base64_encode)
# print(f"step:3 reverse the process string to bites step
3{base64_decode}")


# #utf-8 decoding
# utf_decoding=base64_decode.decode()
# print(f"final output {utf_decoding}")



DECODE THE BINARY TO STRING  :aGkgaSBhbSB2aWNreQ==
sucessfully Binary to string
aGkgaSBhbSB2aWNreQ==
DONE
```

"step by step "

```
import base64
text="hi i am vicky" #original text
#utf-8 bites covertion step_1

utf_code=(text.encode()) #default utf-8
print(f"step: 1  bites==frist_out>{utf_code}")


#bites to string step_2
base64_encode=base64.b64encode(utf_code)
print(f"step:2  bites to string{base64_encode}")


#decode process
base64_decode=base64.b64decode(base64_encode)
```

```
print(f"step:3 reverse the process string to bites step
3{base64_decode}")


#utf-8 decoding
utf_decoding=base64_decode.decode()
print(f"final output: {utf_decoding}")


step: 1  bites==frist_out>b'hi i am vicky'
step:2  bites to stringb'aGkgaSBhbSB2aWNreQ=='
step:3 reverse the process string to bites step 3b'hi i am vicky'
final output: hi i am vicky
```

DECODING PROCESS (BINARY TO ORIGINAL TEXT)

Base64 Decoding Process

Base64 Decoding Process Reverse the process by mapping Base64 characters back to 6-bit binary. Combine the binary groups into 8-bit bytes. Convert the bytes back to ASCII characters or binary data.

```
decode=base64.b64decode(encode.decode())
print(decode.decode()) #simple
print(f"Binary {encode.decode()} to Original text {decode.decode()}")

HI i am vigneshwaran
Binary SEkgaSBhbSB2aWduZXNod2FyYW4g to Original text HI i am
vigneshwaran
```

IMAGE

Base64 Encoding for Images

Base64 is commonly used to encode images into a text format, which is useful for embedding images in **HTML, CSS, JSON, and emails**.

1How Base64 Image Encoding Works

1. **Read the image file as binary data.**
2. **Convert the binary data to Base64 text.**
3. **Use the Base64 string in web pages or APIs.**

---

2 Common Image Formats for Base64

Format	File Extension	MIME Type
JPEG	`.jpg`, `.jpeg`	`image/jpeg`
PNG	`.png`	`image/png`
GIF	`.gif`	`image/gif`
BMP	`.bmp`	`image/bmp`
SVG	`.svg`	`image/svg+xml`
WebP	`.webp`	`image/webp`

3 Convert an Image to Base64 in Python

You can encode an image as Base64 using Python:

```
import base64

# Open an image file in binary mode
with open("image.jpg", "rb") as image_file:
    base64_string = base64.b64encode(image_file.read()).decode('utf-
8')

print(base64_string)  # This is the Base64-encoded string of the image
```

 The output is a long Base64 string.

---

4 Use Base64 Images in HTML

After encoding, the Base64 string can be used directly in **HTML**:

```
<img src="..." />
```

 Replace `/9j/4AAQSkZ...` with the actual Base64 string.

---

5 Decode Base64 Back to an Image

To convert a Base64 string **back to an image**:

```
import base64

base64_string = "..."  # Your Base64 string
image_data = base64.b64decode(base64_string)
```

```
# Save as an image file
with open("output.jpg", "wb") as image_file:
    image_file.write(image_data)
```

---

6 Advantages of Base64 Encoding for Images

Can be embedded directly in HTML, CSS, and JSON
Useful for small images (icons, logos, etc.)
No need for external image files (reduces HTTP requests)

Larger than original image (increases file size by ~33%)
Not efficient for large images

```
import base64
#open an image file in binary mode

with open("OIP.jpg","rb") as image_file: # rb means read binary
    encode=base64.b64encode(image_file.read()).decode()# as string
******read()
print(encode)
```


```
/9j/4AAQSkZJRgABAQAAAQABAAD/
2wBDAAgFBgcGBQgHBgcJCAgJDBMMDAsLDBgREg4THBgdHRsYGxofIywlHyEqIRobJjQnKi
4vMTIxHiU2OjYwOiwwMTD/
2wBDAQgJCQwKDBcMDBcwIBsgMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
AwMDAwMDAwMDAwMDAwMDD/wAARCAC4ARQDASIAAhEBAxEB/
8QAHAAAAgMBAQEBAAAAAAAAAAAABQYDBAcAAgEI/
8QAPhAAAgECBAQEBAMFCAIDAQAAAQIDBBEABRIhBhMxQSJRYXEUMoGRByOhFSRCUrEzYnK
CwdHh8BbxU5KyJf/EABoBAAIDAQEAAAAAAAAAAAAAAAIDAQQFAAb/
xAAtEQACAgEEAQMDBAIDAQAAAAABAgARAwQSITFBEyJRMmGhBRRxkdHwQ1KB4f/
aAAwDAQACEQMRAD8AxiFyGxZWQfxWva49MD2k0t+mPnO9ffFcrZgQxDWaLbk7gbeWL8GZS
JceK4Nxbc+uFynMk0ixwq0juQFVRcsewtjUPw54HWZZ8x4ollpKcMlPHSpcSSySfINtyw6
```

hR7k2FiIx3x5kERn4QyfMarIsuqIbwyM+oKsYMjOx3bxWChU6X9cMuYZrPWS09Hl0U0uSR
kg1cbqi1cq3NlY3IQEAahe5vYNa+CbZPRZHw/JS0yrHTxreRGlYiU32RpLE6T/
Fbdumyi2BOZUUOaVdLWNBW10saj4dTEyRRWJsyR2FjsbE72A6Xw8qzKFupHfEXeIsuk59N
VTaRmkSloWpSQtKoOyIOtvm6+RJx7y2noq5UrVlJo5KU00kTbtr1DSbnoet+24tglTVWVy
yySTGpqpo/
wAtqekppJmHW63A06rbHfa588fKyvzZFjHCuV0lCNNpajMp41kha53K6mIPfYXxDtix98m
TtrmXYOFaKOOEPI9BUNHcMKgQ6hbrb0HWwwT4ObKKbMJo6aeoqauRrNz5SSQCbFCbAgeQ3
774R4eGqKsqpKriziyKtrtW5pZmKoltuq3J37kD0xfyTRNxBlwglEsS1WgSDbUN7f6HGdk
1ATIoRRRPMFiFI2iaTUzcqpYrQiqINuYZVL+2lrWGPjzU8sy86lijYNZZDIFYG3UaSTfEK
PnCT3EqSQ6t4pU3Ueh9PU4C12dZxDmjQ8yaCnUhWYUyKL9/Fve/
a1sanAjOYQ43mMVBAqZtHljg2aao3DpY38N+t7b+
+F3K8jo8xXnVWc5rmURFwYYvh4if8RAJHtitnvFEGRUkE2bUsZrZNRBlYyte+5F/
ce3TCRmH4gZ9nL8rLwYEJtrY6RbGVqAvqklQT8mdz4jrmvDuRZdBNPRZBC7hPzJWZp5WJv
Ygs1tXe1jhaX8KqnMa6mFMKiOlne8tRUOuqFdyTy+5Pyjtf0xfyDNRw/
kyTSSJmFbUya6iUSXCbkKALE7b+9ifTBrgPi6mrM0rJqp5JpkJjRY11GNL9TuDY7W2vgcT
F3G7r8Sz6VpuPcLUn4VcGUlMsL0sryqQedJVuZGPmd7fS1se+IKBcqnoy5/
c1nj5ZQbXuQVI89wR7emCcfE9NU1PLEam521L/
pi3VPT11BLFNFZDs8fWxuLMvtsQcbKMo+mKKEeJ+ZOOMyq240zSmdg/MqmEQXoQxAX/
AExuXBnB2ScJ5eKh41lrAn59XJcsxtuAOy+QGMp4uy+DMuM/2klQBNS1sMTRstg6oqkm/
wDNe+NVynPIM4yt4xLomA8aqbMD5+2KucgOFbozgh2lhC9ZPQ1VE8pp3elYhXLIQCb+vbG
e8e/
hZTVVE9fwuFjqANRpw3gmFug8m8u2NCyPMKeHL48trS5tFpaRwCG7bnzPbFfK6yWKrmobC
ogRiqTA2Ki3Rgf4vbBNjUC1EXc/K7u0MjRyBkZDYqwsVIO498eeYN9/fDZ+OOWLl/
HcxpVNq2FKpkUdGNwdvUrf64SYcuzCVSUppbL1LCw+5xKqWFwof4YpqeszVXrkMlHTATzo
oJMgDABT5Aki/
pfvjYuLeMafh6BYMnpi9RHGkEjSAiLVa+jQCPEAeu1r2xlXBcc+VQvVyQRSTmRSscwDxul
jcsAd7GxHkRfti3KlBUNU1VfVV8FohLTKYw5lmudRLbALsd+u9uowK5wjFb6jNlgGCcwrV
qZZpqiSUTO5KoF1KAT0uTimsy6TqIvfuMWp6SYrHJ8MUiVSRJe+s+pvsRcC22Icuyqvr62
GhoqCokrZmISILYv1Pe1rbknEbwebk7TPVPIrWsRfuMN3A+fT5NmSVFNaSWNX5ZkHhiJUj
UBfe1zscK0+W1eXVc1FWUtRFWRNZ4yt+X9r4I0UUqLHM8XLikYBalrqu9ri+9x526Y4ZFH
IM7bfBlfiugqcrziOJEX96iWZI4jcqW6r/i72/
vDGjfhxllPlUH704evqF1S2a4QX2T6dz6+mAE9MtXl9LFTI71EVQ0rzN8rIQFsgvuosGJP
9MGsto5KfMkQlkGrQG6AAgWOLeDafeIhlriOkuWI9TJL8TVwJMdVRT8wCKcqLguLdQCOhF
wox9q6DmU0snLLhpDIrDZW3G4PdfI+uKmUzLV00szxR19RCQqQNYa7AgDxDr5H6Yt53W0U
KxUE9YtBWsCaeFLjWF8RUC1rWN/
r6Ysg0YPiVpVAYGLToZQw1Hf8Arjse6Th8VFOrwV08cNhoU3NlsCLX7b47DbnT88Z7TRFr
wxmOS26gWBPtgbHl87W1EL6E7411eCKnMZZqmGNXggYqJy6pGz33JY2Fh7/0w15JwOuV5X
P/
AGUUtVVRqtQtPHUytBoOyhgVjBa7FmsLAE36YzGbaLjDQmYfh3+HGa8T1cUtPrpqBXAmri
NgBa4T+Zu3l5na2P0dkXCmV5OtJ8DE7tSQmKOWVy7DUfE3lqPc+w2AtihwNmcGZNmHwSzP
FRzfC88EmBtIHhGqxLC9yQoG+1htha/
EPiGuhzuSih1JBAqkIrEBiRux6ee2EZs4xIHIs+ISJvNS5+JPAmYZ/V/
tCjzqohEMUUVNRGMyxGQMdbFb7Erbpb5dzvbCNmGYScH0k0dTMM5bU0ckdQkiRwkWIVEZh
pF9yLE7W6Y1HgriJqzJJpa9iFpQS87nYKF1Ek+g64QM94w4bk4ilmybL4cyKvzRPUKxXmH
5mRWGx22a3c2FsBv9ZFdTQPcdj07uxVYO4kznjF+HqSqr/
hMmglBZIVl0yOhOxEQG1h6gn0xnVdXsjMYa5yt/EzgLqPna5641XJuH+H8+rp5cyrq/
Mp8wYtFBPMTyWPcaTZutt72A3G2I8y4SyHg5nmphDmEU7hY1rIwY42HViOspG9r6VHkeuD
GBRZ8f3AfC2M0wmWUUWZ1MMlRHLJBTOLNUSiynfoPPtvth3/
Cyieg4ppDVZ+ZA0qvDRxJqFQ9iRck+G3UkXw7ZVw9U8RZPFm9A9NFVyvphaWhSNTGp0ghg

u2wuLAi2wO+GzIOFVo6+ir6mtlmqqQMJAJS8bMVI6EAi1yQPb3wABLAAcfxK7KxMP0zipn
BMbxlogzC9wN9hfEedu1PTEwRGSW6lA24uD1vggsyp2GPbTIbiwI8jvi+e4QMwb8XspzSv
+DrZkAlRqhTCpuWGpSCD6eXXCf8AsylpKKB56ipnqHW7qsulV9LAY/
See5RR5rBDz6WCbkPqUyA3UH5ipH8Ww8+mEWp4O4RlrpC/
xdQSdaQRVBSNhfcA2vc23BPtjO1CuW7AH+8SVxu/
0TLstyqesq4qWmrVSKVjphnXdj1Njtc+X/OHPhJKLJcnaiqCvxYYh9PV2Lbg/
wBB7YvrW8LUE7PS8O0Uc6GweqDSuhHT5rkH7YJ5bxZmmZzyx5bBFFAn9rMUCRxC25LefcA
XOKy51YhByfsJopo86JuegPuYJoP2jU/E8nJcyrob/
ujzU7wBQRYm6ncC22rBHIqDO6ONcupMtzOPkvcVlRKrDSWH5cJJFltfqDbtj7mPEdFly3q
KubNKg30859KNY2Olelxt1ve/
rhXzf8SqqTWkFMsCsCqlnJOnodttzv0t740MeGhbmpnu4VuDf4Ea+IPw5ynMqLRl9SKCse
oNRJUkmY6m6ra4FtgB0sB64Gw/
hulNOklBxU8UxN1LRJuLbkaXGM8rOM80mXQ9Q72O2pzYDpbTci1trfqcQw8YZnFGyNVTzX
Hh1OtlPY2IN7dt/S+HMuJ/
qEWuR1+mazU0nEOVxpzaeDiCLULmkfkyqOxKsSLH0P0wtZv+JKZPmElNFkE8GYNpBima7M
eiiw637WwEy3j7ModCTVGiEkFjZNT77i+9ifoT53wXq+KMsqpYq8U8de9PNzEapgHMTckg
ODcelzhL41C2pNfEbjyKzBXWr8wFxJwPxZxHmEuc53X5Zl9ROo0Us1SVaNB8q2AIH3OEr/
xrPZq6Wko6Kqr5IvnNIhmW19jqAtY40XiTP6KslmFdwg9JqvaZkmikXyJYixPvthcpuK8y
eKkyLIJalizaFhhbeRyevh3JPcn+gwPq0QFHE1hpMTCt1H79fi5FkOW1NHls1Fm1NNQzxT
XBnQqQrD+W24uO3TyOCkWVZhUzgLFLXyaRIssX54RBfTst7L5A26dMe864a4wy+BK+tpkl
0byaKpJnjHe6jy72viSk40zZsrak4eonjlkkMtXJRwMzM3QAlb2AA2GKIX1chN8fbmQ2FV
oIbEqUWRxyw1DRqgqJSsBVW3U6tR22N/Di0vDTyKA0tfJOl+bNK55cI/
lW+7N3JFh2364E0mbV2X5smZOJI6qNrnnR2LA/
MpDeYvjZ46LMZYwWhWqgYX8MaEkHts33xnazLk0xFGwZxxqvBmYw5T8DqQ1OYQTOLOUkYC
Ve9yPLuD9PLFekyOnblRxpDJU8vQ2prlRpBuF7k9rdN8aZPktbU1IkOWU0WkWMs8JOkdz8
9r9+mF2rlOSTyxS5xQ0FErNogpUj+IlF+jCIG1+92HX0wjFqmycDuKIUGR8JZJJW1Y5xYQ
qhVItdxEG6i19uhAHvhizSgEMEUyqCwZQwO2rbrbz3wuvxbl2SVK6Ib1LhTUKpsFF7qNv4
gDv79cGIuMeHc2V4viZqeS10M8fhY3FxcE2PlfHqNG6piCsRfmJyabK3vUcQjw3CtLmjIS
Cju6qhFrkrqXfzBGC09LLPm1NNV0sTtDAZEZdyh1EOhJsSCNG47jtfAaAuuaWuQh5b3Xow
6H9L4vy5rNlUNPVPS1NeY5Ph3gpwDI2o6bgMRexUX3HXGkRKH2lzLRUyUi/mrGVJUq/
UWNh+lsdinDVSlNcQcrJ4/
FbUL+fXfHYORPtA4SGolo6J8zqKVFSDVAUjRmbYJF0jVQLm/
iNx54p5nmqUktXl+a101VM5Qu1MixrGwvcb7FjffYgW7kYN5HAMqkrXr5JI2Tc62JWOIEg
Mx6FnN7WubWHbEtZlWS5lmkxqqSGWWBlWXVsFZlBAI23sV6+eMnNjcKFw1d9mNQLfu5n3h
6joqbLqaqyqmmlapAUtPMAVW5uxHQ236Df23xFnuR5TnarPVFZyoCLURy7ncCxYeu319cW
amsy91NFUOkanwKhk5TAjoE3B1DtbfGacQ5Rx3xLWz0lLTZrTZS04eLmzRwrJGjDS0m4YN
car2udri4vhhpkCvz8wk5bjiN+ZZnw/wxlclNU8tKFZTSTxRxmUqxQtpdBdvEO/e/
rfH53nrqV6+qlpoTS07SsYoWckxJc2Fz5DzxutD+GGc1te9fxBn4SaQgyCjS7ttbeRvTbp
htHAfDsrxTZnQpmlRF8s1aBIfqLaf0wNWAAKAlzeuIEK1k/AmU/
hDWyPm01FPMsPxtJIkDNvrN1uR6HzHW2H/
hzJqLMaZs0lWPNJZFki0zj8mHwlbFe9wSCd9jbDmaCjdUR6WErHsg5Ysvlby+mKGYZVI12
QNVU9j+6zMSASLalOx27g39LHHEELUW+YuSfmfcsSqkjHPq44Vi0g09NpIiIAumvqfsNsW
p5gtwOl8DctqI0+JiXSJEkDOqx8s2KizaetjuAf7p8sWHINjcb+uDx/
SD5lZjzPTzEtjlmPniuSMctvPBWYuEqebAXinh419NJXZV+75lEutSotzbX8JHS/
kfP0xfhcLbfE7VkcMbO8gREGpmY2CjuSewGJKq4phDR2Q2pmJ/iNVTyfh/
k2cGUVFT8U9NqCgyaLMwB72GlTY7i5wkrxDVrlcNGZmSFRr5QNgSTe59Thm/
Eytm5tBRwnQkAYR0qRCOOHUTsFA/lAJJ3u58sZdU1UnxMgmJ1qdN+vTC/
TCHcO418zOmw9Q1+0yG1qbHYk9LHzxBLWvNcuWe+12Pbt3x94e4czjiCNpKClHwynS1TM2
iIN5aj1PoL9d8Xc14Urcjm//p1dG0ChgXpZuadYGy22IuTa5Ftu/

QwxrkxYwsRuA4gaWpbcX6fpiLn+L5zf32+2JaxNcCxoVjaL5EG5PmL+fffAuFpZ5VjhjaS
RjYKouSfbHDnqKhH4ohSXa1t9ztjSOA+CM9z6JaqbTlVC1111MRaWQG99Ee11O25I+uKvB
HDmW5HIuYcQqtRWJaRYCC6RLYXIAU3YEi52A7Xwc4g/
EZzpioJGtpuRoAupGw1G+w6kdyfc4aFUC2i2azQE0igRMop4qObiGprpQioGd0jjjXoC17
ne1gOp8u+Jvisvp6aSWFYZJnUszxUutZADYnWAAfI3P9LYwx+La12BUIgW4HLdlKk2uR4i
BtYbW+uPP/
lE7x2qqyW1xdFjAjYbWXSpF7W76SNjuRbAqUHQh+rl7uaZxfRVlJrrMqpZpqfTqqKWE7sn
UmPruOhAv5jphFzfifijLqRVhyyryTK13jiipWgjX3NhcnuTucT5BxzNTxmPkqgRw6mA20
i+50nqwO99ySTtvgpn/
GVQP3iKrcxaR8RE48AN9iEYk77XB8x54qHFjxAvjvnupq6LWEsEyKL8ExQpOK6rS3xEwni
cWaOoUSKwPW4a+L2V02T5hHM6UxBjAYiOQqFFxfv/
ANvijmHAmd8Q56k3DeRVEVHWqJCTEYYIm/
iAL2GnuB62F7Y0LhX8Kc6y6JBNU0VIVYMbM0zHoTewHlivlwsyWhlzPqkb2sBYgjK/
w6r6yKOoMkFDFILp8SzMzrfY6d7Ajz+2Isy4TnyLQ9fPTGNwdDQ7ozW3FrAhje9saXXZTX
0ETH48zyMtlEFMZJB7KTb7/wDGFjiau4rEbplmWS0KIbcwshlJ89fb/LbrjLQZw9ZRS/
mVBtP0zPOMODc4qq1cwyHKc1qY6hVaVPhHGh7diR4lPX0wLynhLit6tY5sorMvS41T1cDR
xqPO5G59BjWOAabOsnaqr+Ia2okrKxQkdG85k5Sg6i73J8Z6Adh13NsN0efzItimu+xBOx
HkRjdQ49oDThnyqfb4gCOjjoMnouTM8+inMDM24lYDwm3mdRt7emLsCyVdJPIsiOZPzYgE
8UblAQL33GpfuTiau+HrKmGlokio5UUVA0+FW6gg26EHcH0xNldBDlqyGqr4XTlCMJG12J
DHSbgDoDb/
ANY0Uy49vBmc6MWs9mBqetXkIUiEgKg3Km49OmOxfppJaFXXLYlq4JXMut13BOxXbytjsH
6y/
MHYYyZhldZX08LLNHS1FOytEXXmqTcE612v0NrEdb9sfMr4dp6P4hp5TVPUS86SyBFZ9t7
bknYHcnoPLF2gzKnr1JiYBh1VuuLn2xRx5UyjehsGMKFeDKq5Vl6VvxnwNP8AFWtz2jDSW
8tRBOLfzN0x2PLusa3d1QebG2DJCizIq5HDJM8jh6cxqPlYuDq38hidcDavPKGDrLzCOy4
DVnFwVSIIlHq25xnv+oafEKLWftzHLhduhGvFeevpYLh5luOwNzhEquIKqpY65Wt5A2H2x
BHWFmGonGZn/WWIrEv/AKf8Swul/
wCxjFxBFBmTxVcDy09ZCCsc8Zs2k9VPUMp8jcf1xQ+MmijAk8bDqyiwb1tfElNIJYD6EYr
zqN8aGjzvlwh2PJ/zK2TGFYgT02aqOuofTHlc2Rm8JY/
TFN1vjwqjV2xb9QxW2FVr2f5B9zixBRivqYjWOZYhIGWE7RqQdiV/iI/
vXwNg8NsHMrsZYvVgP1wSOSYQUTD56DNM1qamppYoeTDJNTxmR99QY3JAB6nfAeLg2FJ46
niWURUFOCoWKQCWe3YX6D13JvYeeD/
DsvEB4x4iy3I8sFdJJqk3dVSJg5USEsQCT0t3+mJcw/
DrjvMas1GYU9OA3zM1Ymw+3QeQwbPk3H4lvDjwkW5ixnPFVRKvwmWXo6KMcuOGIAWXy2/
X3vvgTDJVT0wp1JkaodVXxWsTsNzbDzU/
hfmaQXvTpHFqYSw3lBbqVK2F7C5BX2ItbF3Lfw7b4YPU5nN8O3gSaCJFaXa5Ckn8sDoehH
c9sIJ3dyxl1Y2bU/iZXmtHNDVmmnK085cro2GkXt0v1P8A3rg/w/
Qw5RTNKkYlnJ0M5Pzb20gi5tfy8u+NIzvhzK6doY6/
LpZJNRWKr03cOANyzHc+Y2BJJFrWwp1cCJIIqYNIEvGixAkML7WGxG3XEPm2UBMN7HBgjN
K+dIliqZJXlUbRs1ggW+5F77fw7L1wuS1LuxN233JJwyZyiQQqfyIHYXIa40gbW0jqB5ee
++Et5r33BF9tPQ/
TDUfeLhIoqWGnGolQNvLscWIUmdlKx6266UYFh62ucDQS0mhAXdiAFUXJJ6D3w+5R+EOfV
lMKmvq6HKxILrHM5MnpcDYYKo1cTN9IgWiocwlqY6daKpd5RqRJIigI7m7WAHmTYYbeHay
bL87oZamSmneKqiK8upWbW2oLvpJubE73xWn4MzXhiJv2qFrqSYnmVNNqkiROW4GoWFjqI
IvtcDcYip2y7L6nLDkVFVSyQxxl5qaq5tTJMv8AayLDZtKhr6b6fl1C4NsAebqAcZVvdP0
sZW1MEBezaSQbjrvf274rtmUULaJ35DiPnFZfDpS9tRN7AXsD5X9cLXCPFuSZ3GaXJ6h4a
iGM6qCo1JJEBsdtwbG1zv133OC2ZRUlXSRitMcbLdAzMGHiWzqSwsykbMpBB26EAhG5geb
EcAD1LGYZvT0aa6qrhiFzcX8Vx6bn/vbChm/
GqPrjoCYxe3Ofdz7fyj7n2wscYilSDMaPLFoS0QSGCs5lxUlNLmFzYmyDoSb/

AMNycXuFuA+Dc9y6GtgqppeYoMkRqncqR1AJsdN+hsPpfCXVq3E/7/
cbjdLoiVXz6lia5qU1dSdW9+
+J4cyaa5p45prEAlIWI+9sO+W8J5DlS6KLL4YhsT3JPqTgitLTLY20HqPUdz/
vhHu8R5yL8RE+Dz+qq5aqiy3mLyViiMkyxhha5N97C5sMBeKI+Isl0yVVG0kEpKq9Opk0m
1yCoBI72O4NjjWPhqhPHBIH3tpAAAHY+/
bH2vpTWZfJHMpicIzK4FypsRb1vexHe+LWAFm2uIh3PYiTwilQmRQfGSfDyuWflv1AJPrj
sR181KkypU5hU0mmNRGsdIHVkts1zfrvt6Y7GsMCV1K/
rNLFNUvC4KGxB2Iwx03EmmAmcXYdxtfCjHJ198dVyFYH048ImTJiN4zU1iiv9UYKzi6Zri
G0Y/ujfAapziSdiZJiT/eOASvK8fMuES9tbGy38r9z7XxDI4/
nZz6bDBlM2bnIxP8AJkhcafSIVesXe8mIHrIu7jAmp5kfLMo5QmNozIdIc37Eke2IFSeRg
EDElgoATv2FrYJdMJJeF/jYP/
kGOXMo1tYnFBcvrUt8RTSDUBbVHYXI2HTr3IwFzLMvhJHijo3nZD4wqsoS1rgnzw5dHu4E
Evt5M0rh6vWeORFPQDF6WxvhD4MzhZZZdMMsYIAszXF7772GG742J162Pkca2kxnHj2HxK
OUgtYnt/
mx4uFbET1K+YtinLXomos4AVbkk2AHnh5MTC0cuC+U1KpOms7Bgf1wn0mZxVcYlp5VkQ7A
qe/
lglTVoRgSRYb45Xow0FwPwTMuX8aZlVzBVMkhR9t9JLEbeQuLY02WspZVvHVsUNiFVwA3r
ftb/TGRcSZqmVVKVVTNczOF8KAEC2xY97dBfHqmzgyMtpSQSO97D/
3iH1D42PFgy4umXKo2nkR+VJK2dKxqcckIGMbTEmY38J03IKm1wD38iMDc5zBI2kio45ZK
id2kR0sGvb5gWIte9jbpgPSZ5WRS3EzfmWLhjcNYbXBwfynPMvqY3Sqo0EzR6xKFBB0km5
Fxa2BGoXKQp4gvoXRdw5/
iVHoqaLL1NVTxVE7i95XuI2VT4FBuQvf6Xub7Ib0QpK2Wnk5rtI7SQylLFkIOwN/Cel/
+caZWZxHDThI0q6wxaVY0lKFCDva4tt9f1thfgyyqmkp2fKmmTmc1ZJ5RddYtv0Y2AFvL6
4a6BuIltOSBxFj/
AMUqMzVYpI45FVtSxNClidrm9tz02/8AWEHjrJpconWnq3NO5GtEfVpYX2KkC3v3FrbY/
QkFHmNPJJGlDTSRyLcMzkPq8rAdP9ftgX+JeQV/
FXBNTQHLYhVw6ZqR+dezg2IubWut1udtxe3XBYcWw9mAdPtFg/
mYV+GqxRZlNmk4DfCKOUCNtbd/
oAfvhnzbiiqqZXL1DkE9L2wqU1HmHDxrMszWllo6pCrmOQWJFjYjzB7EXwPlqXcnxHC82N
smQ88CbehKphB8xwg4hnHg5rhTsy6tmHla+DmU8WU+XRlaSjp6I3vzKVeSw9bgj9bjGZQv
K8ixw3Z3NlUdST0GNb4SyfhrIYY5c8jjzDMGsWaQa0i8wFO1vU7n06YV6W0/
VUtPkGQVtuD834lizNhmNM8NJVeFK2ojiD88I1kNgQdViptcC467DBSP8R6eWDRNMIKmKN
YYppKe7Si5J8IJAI6m53v1OCfE2a8DZhB8O+WyuAwJakIg3B23FtsK8WScE1NUZpavO4Q0
mto5OXKjC+4uAGse9jiyroONwmLk0WUtvRSJe4pFLWZTeaSrInqA89VHGI+axTwMp3uLLb
r2t6YIcN5nl/
DuW0tBlsdTIqPzlqCojcl7XuehBFgR0IA6Wwu51THh3Lqmp4dqlzLIZmC1dFIHCKL7MVJ1
KQbWdT9umFvNKeoFdSjK0Z6WtjEtPqe5AvZlJsN1a6nz2PfAsGPKniHgCISmce7k38ib1l
/
EMdRPCZKoUbOv9lI0bRsd99V9yfK4G3niDLeLKWqzqbh7NgkGcayqtEHME25CtGxvYkDdT
9C2Mrjp8y4fyCumzXn0oniCU5iKgNJqU23J6AG+3Tr1tgblPF0j8RUstKgjKyqkIiQFlBA
DAAWHW/Sw+18AMKsCWlXKAuTbjNgz9HWejhkmrJ4aeAEEyStta/
v1OEzPuO6Wsinosikqfh1B5tTEeW01rnRETexP8xG+4FiMSZ7Bw3m1NTxcVxhl20zPMUkQ
+d0IwsRZPlUGZu+Q5vLPSi1+cn5gGoXAcWuCBvceRxbwY8acA8n5g5NPlFmrA+0LDM8xpi
fhIRHFMFnEdRHd01qGsdwL73Nha5OOxNl/
CMuYUMTRMWWEvDd7sfDI1hc+QsPpjsaNSlcT+FeLqfM1CS3imYEhW/
iW9rj0w1SvB+zZKmpXmK7cqKIGwle1yTax0gEXt1uB3xiWciRakOn5M3hkCx7CP/
41HsLsff1xodNX103A/CeYU80btTzzU9SXS4MplvuNuq+3THnH/
TlZi6/1L4zHhTJauqZ51580UZ2VQ5CKo7AAdB5AYKVNAUpkHxMs88ukRLSIpRrkktrZTqU
AbkWIvsLjH2eRKqslq8syymimkqhzGkdTJGQdkhJW+oqQAq9CNW+Bb1Eccc8F+SY5+XLK0

YkkEaMxWKQBtWs2LWUDZdyN8CmEL95ZEIOiZbHV1OYSRx0dQqw3nTQWTSCFCsDzmO/
kAADc74t1mZDLYzVzPGjCMRzFpiDGqg3YKtwzKCGOkHZSO1sA1rJ4KyetpZa5xNEzwVFPR
NzNEgIM5MhIjuT6bCw67Q5Bm1PWT1NRC9FC0paOPnLpZmK2a26hmPzBtt7XDHDNpHPiFC+
ZVvOklK1Mj8zQ4ka6BkbodWm1m8Ng2ykgCw6ivij8PHC1L+76j42tDIV1lbgm3MI8ZJUg3
/htiq9KlFFI0LVEtPyxIKkVSSIX1KGJjCArqa40G/
pYWGPNTTfAzjk1QhlaJBy0ksTKHGuI6wCrDwsAR6X2x3kzvEuVNfF8WooKwajNbmyrYyhQ
L3AGwI3BFh12NrYkbOfzyQ4kB3BKFQ47Gxsd8A5EkkcxZagkqizLOIYCpQ3OlbX06fD8xA
63G18X14VzapgiqEEcdWVHME1QSrLv4dIBsR1uDbtbvgRkTF9ZqJyIG/
mFKyseONJATyZl1IevfcX8x/
3rgDmFZLU3iiJJbsO4ww02Q1b8PJQ1sqpUpMZg6DWFBWxW+3ob4oQcMCkn501ZJrDXUhdG
k9jffBtlRxaG5nO23gwLkU1fltWx5cklPLbUqrcg9jbz88N0c71ET8uWDmjYRNLpdj7HEe
Sz5bw3xBBNmEkqRzhommZrrGXtZzY/L2Ple/
bGkvlsM37xDHE+pQwsbpILbHv16hh+o2weNSwsRYysvUznN8nOaRx09ZNHSt8peVTZiD1A
t1t6j9MLtNkmY0FW1PQRVGYUyG6zxpcDfdbenbrjZP2flUsPLeISR6rtFPKfA9723JAt26
frgZm9ETLJNlVYyTFSqxNIeWxt0JBBt5dbde2HnGGHuMamrdDYAmex17JKyOjI6MAyOLEf
TB3h6pSmqY6lDr8Wk6hcKvcW+
+Mtlzato+JKiHNomgqFJilibYqQTt7b7H1w6cNVJMRCFdBUEFj1F/
PFHLiOM2JvabN6ye6a3PWU02XySUWuotcKqpbxb27fbASkWV5I5KYyTG1yrIUKgHol++/
1/TF/g7M45qV6NJBeMl1JOkt5n2udsF2pqiTVofWrXBDG4t/
iHQ4v4z6qhgZXJ9JipH9yBUzCWN+XC1PLGbxyNIAr367WNvqMSUk1ZHE/
xFHUxkdX5iyazfqAtunbb6Yuw0zcs8uZgCLFCdajzGPa6ujpEkt76RJcMPPTYf8Ae+LQB+
ZVLjqom/
iDw3TcVZJ8NUxSQZlTKxpKiRQCDb5SQSNLWsR269Rj8y1UclLUywTo0csbFWRhYqQdxbH6
/qXljcCF0SQm+hmbS48wSDY4wH8X+FTBxXV1NNCYoqu1THb5WLDxgEbfNq+/
rhT0DZjcblRSxGyCcRZvDKbHl3cX8wDbB9825juXk398K0UE9LVIxB2Nj7d8SO51HfocV8
uMOQZoaXUFVIPdwvJmZ1GxxNTZl/
M+AKM0jhEBZmNgoF7nsMaxwd+HGU0WXpm3HFRYOLx0SSWuP7xG59hbAft1Ilg6k+OYsUmc
FLp86zAxyJa4ZWFiLYg4HzuOCOXLK6nmq4Y5DLGsbFXhYbFgbG22xvt0v0GNMmquACop4u
FqWWO9tQTS1vMN1v8AXC/
xPwq2T1Uec8NRSS5XVHkz02Y2dURxY2l+YL0G5uDY3OGY1QKQGuZ2uGVwHKkV5izxfnEmb
1lNHUBoaaNSqRar2A677bnv7YoUQpKJxPTAJIOjDqMF8zyGlzCiMmX5xDP8LIQZY6Wd0Re
gDOqEX6C42wGkyb4akeSfMUqJluTBRLzCqj+JibWHsG9SMCcbMO4vSanGot/
qluXOHq5R8RIzkd2a+L+V5gYJFljfQ6Ndbdz7YUaeqpI5bmF5V/
vSlT136YvwTNI37vGSXbTHGDckk7AefYYUcZXqa41Acc9T9AUPENNlkCszOorlSsUbH+0R
Sep8747BjIMj05PS00opalqWNYOZI1rgDbaxtsRf1vjsbAJqeaJSzMIzDJJZ6mqNNasqTO
YEcNqWWZhd2DbXAGwP+
+DXA+bUeTzy5TWyiTJ66QxQuoA0PFYCYHtdvlO+6jzwQzJ46CF5qCNpIY4Wp6GIoAzXW80
unsdNgPLWB/
DjLs34lMqyRwwwxq420LvY32vf1OBCC7imPFTYuOcszOSjqKmOVZzcn4qNSJIYtNiSSWIY
/
KdI2ve4wkKGoYIpIJXLLpkinp2Oia910BiA9yxIYsL72uFIOBnBn4lZlk8MdNXtJU00Z/
LdWtLEPIE9R6Hy640ii4h4b4oWTmJQ1dRULolYn4aqIsAQWBBNgAB16emEHTg3tMcuoIrc
IlR0tPmFJm6ySqJRMr1E0MrQ8tipUKWbYqBqvtufIb46nq6usn0QMkiU41UkMpAFx1hjVQ
SG0mxCkEsBvsRhireDsghlkloxV0gM3xCxMVnjVrWHWzEDcgE98BZ8ky2jt8HXVsDKoB0w
soNr2uL9Nze1vQi2FNp3HFXGLnQ83K8M9HTS8oxqTTKDBJQ1Np5VtsHLAhSuo32Gkr2tfH
umyw1Na8dZRXBL3q9OiOYWsHVSLl738V7Hra3WtU0eW/
E1B+PmSCWTX8OtECF8Nha52IufvhlotYpo9dTU1A+fmTm7MSBuRf26YpakNgS/
mMGVW6l2ggjpqbTGdCIASzG97Dqzd/Un/
TFWXiqnomdDHKSSdIWxNr7XBIsfPATNK2fN66Choo5ADOF8bAIxv1Iv9B1w5S8GZflvDM0

2YBaiqiRpTKRYA22AG23TGeunFbsvJMWXvqWcnzWnr4/
yZFk7G3b0wQgppap5IoUV3jAIBvdl89jjOfw/
eY5lKG8asQb2ttY2BHmMNHGEGXyxQPnEs1PTxSB+ZDIUYNYgAMOxucAmL0M+0HgxOQbkJM
953w1U1kbJJGvntHYKPc4XsjqeI+G634fh6pTN6eJvzqCFhLGovci4uFY720336jfFnLuK
+GIY1jhjq6hksqKzli57Wuepxeqc/
RM0pik1fBTItpKWnqBoJP8AFv1I9LY1ArJyDKAbxUdMq4lyPPI5P2fMqTR3Sqp5zpkpWBI
s6+QNxcX/AEwFbiemoqmtNTVKIFIXUgSSPUOosTY22v0622xdjSdcnnk/
a1PDBO3MWaKmCSEW35oUkF/
U2v3GF6vqOEKqCly6qr3r6pKgFTVU5dd7ggLsoJ9trd7YtspcAk1UC+eIH4xoMh41+DzCK
sqI6hIzFHLFTX5ygkhm1EGwsQLE7DbbAzKaBsnCxVWZ0ciA3VRzFc+mnTbv5/
XDm1NQ1WkRRiGEDSADYqLbD6dsJ/
GmTT8xzTBzywhAbqpKg4o73Y7T1NDT5ihsGOPClfHS10MkyqUclXUi4023FsaCr002maip
kqFG1kkMTKO5t3A9MY3w5O1Tl6RohMwXSVH85vtf03J98OdEVmpoKehDLWRAtLMXI19gR7
kG1sDiyNiYrXE28mMZgHB/
3+43z53StEVh5csyi7Qnci3cHvb0xTbiJJfzYOXJEp0skikNGPPUCdvphYrKppc6kqLqIo
bO7obXK9d/Mnb6+uKcdRHDBNUlrNMDGu/a/
iNvIbD6+mHnVsTIXSKBHA8SOmtopI6mnBs/VHS/+/
Y9MD89ji4iy16eohSYAkwVIARo332I8z0IHXAGGpiVFpYQXnkOqRRuf7o9O5+uCkdTzNNO
KdYjDuAZbkgndid7D1wP7hmsEwm06qbAmVcQcOsitMlhY2IPS/
bCPXU0iOxCm46j088P34o51WZTxVM7RwTQVVP+6TFDdYWBDINwDY6luQTbpa+KHC2US5zk
AqpYZComaOORlJJUAb37gEkfT0xa2jGgcG5RDlnK1RgXgSJP2lLWygWpI7oD/
OxsPsNR+2GTMMyqa1hJNK0mkWW5uAOwwPrstlydZTHDLpmCklUJAsT3t64oQ5kUhcKw8Qt
54q5byNfibGkYY8f3hSGtKNuTgrRZzVtSy0jVLvSzjTJC5urDyIwlyVR3JPfFmirAGBU4W
MZHIlk5Q3DcwnnFTmVPQtQUs9XHk4jAMSMSqi+6vuBa+4JG+198K9ZJTU9XzIZarUPFHay
6V7C9z2w6U1YDur2Yrpa+4YW3FsK/
FeXxpCtRSkBVYAIP4Va5+wIP3GLuFiTtaYOq0qYT6idGVpXy6q1fDxVUdQWW15BIGGwPhC
g6r9saH+GvDa00q5xmpXmQqzUtPcXDgbO2x3A3C7+fpjNMimno6lZ4riS+xsb2t/
37Y0rIqx6eWBIdWkFlJJJABY6Ta/
pt1xdx41Jsym2dtu1ZrmXQLmVNz5amSnkDMjKkpsQCdLduq6TjsKGUZv8AsqgSFaGucuS7
GmBKA3KkfNsRptbHYtVKcAZ/
yaZVrBJVgvE0UCM9mipEazsSAfG7Ei+4uSBfTjJM3pJJMwmkWnWDXIx5aCyrudgPIdPpjZ
uIKn9pM9dDFLErRJLBSMCVp6cXEQIv8xIaXz8PlhVrspjhjkklicBL6xcsVK7EDzII0jzI
vte2KrNQuPC3M9go3DEMDZQCT2BPT/U/TFlqXlSBCD4QSx6adrn7D9T640n/
AMUloaROfTqKiIa50PVpWIGjr0DFEHnoc98C80yB6PmO5EhW/XcuVazAD+9JpX/
KcKDGrhFfEUYc4zWlVtFdOijYqWLAdyADfYCw+uPLZ9mDk86RXIG5ZAdz/
sMGs5ySeniLzRBdDESEDYkG7G9umrb6YDPlrabyKyg3JJXrvv8AbbBjIYBSUqjM6h2bddx
bZB/
TGycITLVcO0ciWctTqDY2BIsCPPqMZNHlMjNYRm9gQe19v9x98PHAOZfsqNaOtIED3lic2
CqpOkAnsCQSD64oa5TkxWvYhoKMizaF4c9XlRNpeRXbS9twTc2t9DjRuIuLaGo4TmkopUk
dyITG3zL1ubfS18D8y4eo84j1mwfqG8ttttv+++FjNOFZoWSKGS0KkXu25FjYA+ew+/
riimVMgUMaqFyJPwPHVNXVEoQCB9LI6AeLY7bdulsS/
ijLLVZbDQwMNckgY37KoN9vcjBeimpcky12kkSOKMamc7AbbbefpjK+IeIZs2zuSqUtHHY
LEh7KDt9T1PvicSnNm9QDgQHO1ahXJcvioVMhm11JHUbADuB64J0FasLSzVKJK8ZAiRxdS
e7np07DzPpgbl8NO8K8y+oLY6hck+4GJ2QuuhI2FjewG9re2LbmUzPUmc1bykpUOjHa6D5
R5Dyv6YBSvNR5pT1yGR+TIJHXoWF99va+LMsxp2YOBYefb198csVQ9JI6UtQVWzM/
JYgDsSbbDDEBqxJAqaNldZHWwLWUpEilb2DWDD/
cYtVeYQOzxOC7yquouei6BYXPpscIXD3Oy+fmUDyaXYFob2DLfc2vtt1ODuczomdrUQR0l
dSx2sjS25pCgAEEWKixJ37WNr4SuJmcAdfMdjw5HBKKTXxPuV1MeW5tW/
CWCIFkDMAVUtfoehwdyivWjrhVPUpVVUnhWNdxfsSewHYYTs24noEzQ1tTlk0lXqDKJ5CV

Vd7ncnYg2AHht74DZFnj1FXIAQiM91QbAAnYD0wzVaXYu4Nc1tFn/
wCJxU16rp4NO3OmSS5SKI2MrA7tfso7ffviBoYRL/
Z00fJs1RUKoZYRbZVve5vgWvEUTcyNJCiN/
bTt81v5UHYf99MD6viSkSmeQoqU0bWhgZvCzd2buf8AvtigteJqWR2Ye505ljEZVJJ11Qw
k2OmxvLKQB7jE+X6DFKRKRTUwMkkh3MjXtqYep2Vb9vIE4zit4xZ6eWRXeWSZgZ5I0JAF9
lvt162wz/
hrmMfEkNfktVI2WiWSCSOVzYyRqWDrq7Mb3HucE2PJV1K+TOqjg8y1n0FRnawfsaCFJYJS
hqp4kkkW43AZ7gEn+W3TtivDl9TkcRl4hr2lWQbiVzK9/QAH/
QY1KPhpMrVY8seCNQoC8+Dm23Jup1La/
fr0xZWgrkvd8unJO94njP31NjQw6RzjAyNX2mVk1K79yifmvOZ6ubMpaymhmpIR4Yyo0Ej
sTY9cfKWtpqiQRZ3Fz47WaWMATxjzDfxEdbNcHpte+NK/
FqmqoK+lqJqWOOi5EjsupGjLJ1N7AknUoF8KWWcENnGUR5g1dDlrqAZZJQzRsndksCSwH8
IuD5rbE5AqnbfIhY8rVZifxTlkuSZtLQTSLLoCvHMgsssbDUjj0IIPp07YFpMyHwk4O/
iXmcFbxTJHRo0dLRQQ0cCv85REABb1PU4WNY88Gq2oMspqOOTDEFe5UC5FsSS1LVNFUR2J
syL02BBJO+BMMhX5ep6E+
+DGUxSTLJGQdIALEHcDubf1wS46O6I1GfeNonZbTE3DDbTf2F+vsO+HuihkSFC5T8ttLXG
3VST02O+xt3xX4Ty2Hmq6PsyyKztGCVDDYg+oLD7eeG34CGWMRx7q66tIBJuq6CN99Vo7n
/
pxdxCUGlzhdoZKWpGZyhJlqGC+LTdSqsD1369cdhZ4iiq8szIwxSIFaNHF6nlH5QOlj5dc
dh1xdRlzeGRHlT4kTycwyVTwxeHWdNlXpcgKLAdAnbUMQR01IKyCnplaOloGVyZnuDUEko
GPSyWeVu3gUd8D6apqqZYY69Y3elR5dcMokZnYkF773PRR0GwsN74bspypaOOSGtlWTkxs
ZwDbmP4dYC2tpBEUQ/
wt7YoMNzbZaUhRcrCj5LLydWlSqxK51HnNshPTdIwXYnfU7YjgyxKycVDx8uEqrxJpFhGt
xGL+ZJaQ2/qcHp4Wi0a0TnOWTSgvqY7zN7hfCD/
et3xPArcoyXUPWHZwbFQb2Vdj0Xf3PphhAi7MU63h6CuWCOJNaahqLC4cAjSv1br7HAvNs
qoKaBCiAi7EN5hWOo9rkt/+fXD9l0IMUZp0UQBbqW2K7WQ39Bc/
XAHPUgk5uzCEBVUslvDu1rdltb67dThbAVcNT4mcSZfG7MWkR1Zd3JsFGk6rdPlXU1/
MpgJm9qaAkxKk76ZWiIvylIPKTp1AuxGNPjymCHesjSOBVIlRVv4EYNIAN/CW0RD/
AAHywscQZJUVmaMJUJLH4iVugDONR2v0VNKgf3sCooWYRFxCouKM5ynSlLUExW2jfxL323
/0xcn/ABDzadbTUsD7WswNv/
1iPMMncSyjluDC5UbbA97foMUqfLXmjMpsEUbsEtf179t/
rhJw4mO4qIB3DqeJ8xzDPKmNa+YhL+CJTZR62weh4XpnonleWSR1YXWPuCQNIPnv/
pj3TcOxPwu0tBTVMmdCpvM8wYKsZHgRegJIIYnqNh3tj7Dk2c5erHMo5BC6jUYAXVRc2DF
QSDdb/bEkUPYKAiMiN3cJw8I5pSy/
u0sc1Mq6gZvy3Ufy3sbm3fbA+szJqNo6memroVbo7x2Fr7EEE4Ysr4lymmaMuY55441Uzu
15FuNwSSdyd7jrsLY7PaY8QaKemoJ5JghcAwmIKv8AMWcAAbbDr/
XCyAe5WXeWoiJvEFZS8hamlCi7AjSNrdiT3OIaao4jSikzOClzBKSYBWqVjZY5RbcajYG9
vXpgrn3Ck9NklNRmmNHWQSTSVJeS0VRHpEqgG9uYo1KVHp74p/
8Am1a6pS5lprKWLcRSKPCbet+lt/
t0xe0wAUqTUeUKiUcpzSappBAYxzS1tV7EgdATfoMXGqZYzypmi1ejWPvb/
wBE+tsB6zOFqKmR4adYYioVQpv0HX9cRLWsbIo0MNiQdyPLAOvNCej0WoXHiCg1Luc/
vUQp3lC6GurHpbuP+2x7jjaGiijSsp7ofC1vEvpe3THUVHSSxwz5xJOlPPG5jaI+JmXYKN
iLk9L7Wv3x8ocmjmFLM2cRJEz6JYZZuXJH4TuR/KbWv6jzwluRV/
iU8+TdlLgcyzHNK0AD5lCi3sfPFbk5cZCamtlnZbWCqTt33P6YrS5KjxlhnMat3jnVkJNj
tbffa2IqCgRWcVNO1QCwUEuyaTvtcDc4WuIUSD+KiznY8EQxR5nllE8yUFLLM0qFHQrcMO
/
hF+mxB7HHZTndfT5krLAiRQlRLDqETkHsNRBJFr7d7eeLOSZUtPmHOopKvLQF0ScmW7N1/
isNjY3GLFVw9LDOk37VpaieQgoJd5WY+hvvtjjjQd+fmBvZv/
k1nh7i+lggjTNpZaeE7rV0rflOP7ybqD52A9sP2UV1FXxCSirYqtTuGVhe3sMYDw1KuXxq

afTGszhZUdLxOWksWKnY2UEA7YaJqDh6PMKiOLJTIKZlDPSVllcsAQQtjtYjocUW1WbAdn
Y/iQcat7hNM4qoKGsy4ftCOjcQnXH8RAswVrdQrEC/
vfCBxLNUUOUzz1IIhUflGoAVpyBcAKoAVBtcKLepvjsrz2kp1UZHw7DTylkAqKiXnuAxsW
F7bi1+vbCTn81bX8UZ78bUyVJeip5yS9wAUFwOgC6mJsPPCqyarIDkNAc0L5/
M5PYKEQs1yqd55Xm1PO7lnLCxvvc4oR0BC9OgN+/
tjQ3pajM8yZGjUNBCHIVidl8DHp1PW3+2BddkMlHQmorHVDIG5UFvzGj8X5h8l2277dsbC
ZRQBgslciAIKI6ZHQdBYjqTtvhq4apf7eZYl3iQFiLhXLDSdwenl6HHjJcvpeXWJNVNS19
NHKximSyTJoOyG3zAENY9QD5YauHMsWhy2oepkEgamSW0fS4Etlvt2Fj9L4s46YxLcCEuG
aKJI0rqvSiRVZgkVfGrgknYD1GkHyJ88MlXRLBUh2HOjlUQpoW2tV1aG9DZwCd+g6dMT5T
AQslM+l6lAsqhdgCSCvltYgn3OLebtqijO6FZCEa1zG3huNvI2v6Ni11FkypQ5Xl9RTJJU
UhnBA5bkXOm3qNt77Y7FrKJhFRCGSeSFo3cFFGwuxPcnzx2CgRZ4Uy8SVfxM0hCwATMZIw
CZrkRKbdSDdyLdFHrhvWM6nkSxtHrsWuWbflr9SdR9X9MVMsyxY5Vo5nDy08h+Imiey85l
/
MNvJU0ovlrODkEaqvMChJncSNbcXNwov6Dew8sVEU1bdmOY80JR+GlqZzE7q6uggvaxZQd
Uj38ybge3rjpdD1z08IYmICJ2PRdfifftZbA/b1wQ5i0/
OqCbrBG2mO3YE727ljt9MRZJRslIhkHjJ5kpU31P/HvfzuPp6Y4/
E4fM+zoEyu7QeKYBQi+EA2NgR5ABQftgDSUrLJITDreNtKktvNKWGknrtcg+lmwxZhMoqV
jLgFVKhQLhjfpb+pwMo0kQrapbUoCgsu7yuSqNb0s7e59MCRfEIHzPDUdLS0SIoCQKLSOT
cmCElifdn39bnzxXpIY5qSozKrpjzirExCxYv8AMUB23vZf8vpgpXSrHItJCA4/
vX8UUW7fQuVW+19/LFVaaeKOOnjfQIJCC9rK7jqevymRtR/
w4IrfcgGL1Tw2k0YEscKTyXZmQfM5vc+2ok/5cdV8K0VRAtOsIhpgEkYW3bULlSduiqg/
z4aKbLHqNUrynTMQFZWsVi99tyAf/
sfPENTomzJuS7Sc8ALGuyrGDZ2Lb7k7Dvt6YgIJO6CP2OGlgp3MgJ1VMrqbA7+EEeQVR/
8AXFyghEUFbVQRQpzbBbvcE6RYX3sPlJH+
+L+hnqZpHcjnNyU07FR2I+zH64vQUcEapToCgRiW373Ba/
qTfDAIsmCYcmiWRSKWNEhm58rovikZFNibDrfcf4cfaJKqTmS1QRXmIdvCSI4VvcA23Ym+
58sDOJeHp814m5kVfWUiokcZ5MxALDUWJG/mB0/
h8zg7L42nAvpZ1o42cC5OvQ5+vjIP+2Fe4sQwFeIXAFiZ/
wDipw7mWejh2hgpzNBHMtVmEmoeAyvpGxO+wfYX6eWMrzPhTMcyzuqSgppampeSSXQm/
hW5Y/qAPO/
rjZcxjfOvx1WLmN8LlOW6mRWIQvZgtx6GRj9MX82yuVFkrcsqxQVEVUzRTiIOUW+hgVNtS
lRax879sCdxBKwuOLmJJkEOZ0Lw0lNHlkkBBkmapOmQG/
hCMbk7XvYW7nFKt4fpMv5EE9ZNJJOpaSVIQ6xKGsdg179bg27Y1yPhs5fm65tT0xq56KNQ
YAVDSq8Z1Ib7bgm19vviPMuFZc1zCKur6KKkmqoo0NLBJru17M97DYixB9fPCduTfXiNDL
t47iCmVUcPwkcZzKbLoWUSu0d5FGo6uWguQLk2HqThzoMgpZJVr/
2RM9NOrGOlWAPIUfZNh0tYE+XrbDDV5JmFFTSS5HQ0VRUF2YpUGylbCwAtuTfqcFsgpM/
lnk/
bElJToqhVFJGLMm+pRcAhjbrv6DHPiJO0WeO+KkBxVxXn4Py187WlqESfU+hYxLoDtoJ0K
3Yna/
T3F74FxnN4IaqlyPh6ny+miBqKj42QOsqAlSpS5G56nc7XFrXxoc+Q5fFSQ0McbQR3/
KaByJE8UdirX+YG2/
Xf1wCzLL8zinSoySpjpCt1bnlSkiEElSrEbk2326+uJfEatiOK80LnK/
gRTXJJfFU8pYopNaqyqTHrUSsQCb3ta2/
Qe+L9NS5fTuaqvgDVLRFKSNIyXmlYsAosDa2kntsL9sHKZc4iYZbFDCcuM3NEshPM1Hm+F
R0AFgb+tsenfNqaRqWmgtTVMnJirKZQ1RTHmMoLKxsyjr5jX02weVTsNDmCpG7kxSpqOmy
qQJnYmpYAAFax1LMzPyxYA3BKkG/
mPfEzUT5JmE+WQxRpDTaEblpYE6QQbXPi3v3wwtBXwZPFI1JDm05l0quYy3JsJiGIUeI7H
byN7Yiy2gnWmNZn8lBQ1VWwdadWWMnwjSoXuOo+mENi9w3dH+r+0K7XiLFFX5dR8s1VbPD
KEhYxpRtID8wF223Pa3li3SZfSy8a1U1TM163KIZ1iaKylAVQHr11CxUj1vhl/

ZSq1NMNPJMyOQNyqrTSNuLeZ2/5xA8IpfxcoY5AUilyERBmUEFlqDcDyscF+32tdzg4rqD
aTh15MwdwxpyiPJJpYrzY9bDRsRffSxB229cQz5DEmaVhSON9OrUFX5h4hYjyJsB7nGg1O
WwTSUGmMfl1DE6WN2HKkbSfNSbGx22GKldF8PXVM0Rj8bSDRawkA5gAPXpYW9QPbDceAqS
TIbJuHEUnywTzzR1SJeOMx8xhuSYZuhv/
ACgA9Olu2CEdEkfI0gPG6vEiDawu9jawuQWKkemCs9KY2mik5oMUjXmY3IUxOy9BuBr26d
PTENTBHU0U3LqFm+HmksRcamO4F7dfEL28/
TFpVIiSbnvhmaUZhAaxPzngjXUx3bQSp7dTcYZM9NOkFNI4EYVi5BGzAqQdx9L3wm5JFP8
AtZEkicJGw8DOSepuwP67+fph0ztF/
ZbaBzFKkCzdNxuR5YM8GB3KtJGhjOoI4v4STbw222x2POX8utpEmaRIGI3DIxJPW/X1/
THYOxAoz7l1MYrl5TIbnUw3J3LOxPe5/
oMXLPLC1QgOt2GkHopbYbnyHS2PFJIpgckKFPhKow0qi9/
qdhiXlMzRaxaWEFm8lZl3J9FW9r9yMJPxGz5VzI9dDQwjW5Anl7BVB0xjYdS1yB6E4Jraj
pD4RZQALC29+tvUnArJaWSarqal6jWZW5p0iwU6QIwN+gX73vgrVhXVw4BCixuf4f8Ane3
vhf3hH4lN1Z1YFBqZhpNuo23t53P6Yq0XI50dT4hYfEAFDuTeOIX9FBIHrfFid5zcIjEDS
nhFt2PW9ugGpj9PPHpUPxLBSApBkAIuCq2VBa/
n0/5xwE4yu9hm1ooDIyxHSoNtSxkBR32MjEj/
AA37YljdWfk2URwCxc9LEkX9b2ZvYjzxJBBHzJJkcXlOgup6KlwAP8xYnHipEFBTFNegsN
tb320jw3/wg/fByJa5iiDWpKGawW/
UX6D6Cw+uKfIalpJJIWDvLpjjF9CqBYA+4uW9zi2yS8+mDpcxRNK2+wa1gO/
m3/1xWu0fJSpdSIFUyb7GRjsB/
piZFz5BApljFi6U1iDfbXbbbzAP6nE8GkRPMxFixOoj13OOhEsVMgdzJO51MWFrknp7DHV
xVoJKZJeSfDGStiQXOkHfv5e2OJkQS9UIfiKwx+NBrGlbkuRsL+5AI9cW8thh1QkgJFBqq
WLbliAd/
uzHH3NqBauFKdJOTDHJrZFG7EbqL9jexOBnG0y5TwhmVSsknxMlL8KgDbaiSb//
AKufTAMaEMC4M4Kp3etz7O6qN45q2oWLS6WIVUB79iXaxwTjVXymolhSQhn8EbAEqdQUDr
52P1x84QllzLhSkqqm8U1TT/ESKuwUEXUAm+1rYu0lM0WXpECAElChg25IPW++5N/
tiEFKJzH3XJmpilTUyCFbmJALg2O2kC/fHT0cbVOtixaFRbbYkWIN/
wDKML+RZbxBkmZJF8alZk0s5UrKfzYvEdJB79N+hw1Qs0skhMYsGOlja/y/
8/8Ab4lCzD3CpBoHg3B2S1UlbCvNpPh549pUKkjVtZlNt1IH6Yhlz7KsraGmzKoek1raKe
dLRs2kXGrswuL388T5pBmEzQ09LUpT0rEiolC3lZL/
ACqb+G+9zv6YJT0VO8UNNNGkitEwKuLhhYA7G/
p1wNNW0HkeZNrdmD543nkWN5OXCXRi6nckSIQPY9DhUznhqjrKuCozIzTRU6RsKeZ9SEaS
tve5BPsDh2ZY1qXplj3ijibSBYKuo2+2nAPNYWNMohcSSs5SzG1xy91BHQgWw8AHgwLrkS
eFIzEYok8Gp1XTsALPYd+v+uKkYnp50eCmksZQrAMLqdZ1NbyF8WPiKD4lo/
2nSoUZVkTXpNrPdTfp8y9LYtUCLDIoezlptIFth0/1ucQrq10bqcQR2JBU0TtHE/
LCMjBhpYk6tEgPTr1IHucCv/
FcsGbV+bP+fUjmN+axYQDlabAHpY3+59sNVan7yI9rAgMNJOxVu46dOv8AvirUpohq3SO5
5UhChfm62F/oL4ggHucCR1KLQyRUxSGRU8J1OwuVK0ygED3OAHEEqR/
iXwxI7mJVo51uw6s0wFvrY4ZZdCxzE3HiJIO4JMaAgD09fLCv+J85oOIOGKmOhFU0U04d2
W4SPUq3sN7hmVh298Q44uSDQjnTyI8jSJHdqdxEg3Gq6oCT7FiPpiOtjdJ4yoFjuJGtYAt
uCLdfFcY7IpxULBKkqlagKylRa58Ntr9wMW4XpzIsfMJk0BivUdVufcW39/XBAwbi/
HzpcyqEtrRZH1O4IDEIiCxvvfURf0OPme0fwGRZi9MJIWjZpCV3K3Ju1rne32v2wQXU1Y9
RDF/
Zklrn5SGZunmdhiKWjln4b5Th0kkjQKhfUGY2JDHy63wYkT7kmXhZYJhd0SAIQSGZgOhv5
264OSqr0koQDwxbahtex7WwPyfVQRsj3SOJdm+ZlG/
X9fti7TVLLRRu9iZowRbt4b3PTbEEczrijV0tWkxWnEZjF9OsWPU+mOwz0KBo3MrICXJAP
UDHYmhO3QL8STUokCcujhIflBfE6L8qDfYlz9l9cXK16l6aKma6T1e0zRndbC8h1bbW0qP
c47HYUfMdD9NDyKQ8wKjXux7XtiCdysWuIHxEbEW9tvTb7Y7HYiCJVcSQ00zcs6mUAKBuX
kIAHXoFAv7nEktSYVaRLACTRrAvZU2/

```
VzjsdjlnGdVxrFSRU8OpAsghVgtyLGxP6kn2xXWJa3MFm164YjqAKCyg9AO99KLf/
FjsdifE4S/
HJzowfEC5JNxY2U4gWlWbQW07yc5gF626AenTHY7EwTL0aMsru1rKAPr1JxBOI+fAJD4gT
OwA62BVb+xJP0x2OxxkSvHMJp20IdLHwkb3A2vf13wq/icZ6+kpMphJ/eSQ4Xa+siJN/
wDM5+mOx2Ev3GpGuOigy7KXihjIhjjWAAb2ULpA+2K0dM8dMt3ACyNJcC5072HXrvvjsdh
wizLNNaeNOSCV+I1Eg3Fhff79cTQqHjaXtctbta4v/
THY7HTjPSwCWPWN1K6fS1jb+uPkokNSundVi6W3vr2/
THY7HCRI2Lc93a4EiKltNyp8ZJJ8rW/
6cUamN9RSaNbKX8XSwKkX9yOuOx2DEiLUPA2USZguZVlGslRLICNBspIaVlB877X/
AMI7bYYKCNDLEVk1tzVYkLve6g3Pc3647HYigOoRhdr8yR31ala4AFtvFYfTFd6Z0gl5zk
7SMNraVN7D6Y7HYiRBK6TV1ATRtIzOPI6IwBa3kN8CuO6eGpkyeWsJ0QzVdiRfqhKMR12K
A7dNr7Y7HYhupM8cBySQ0VLFLVR1c08glBZLaQrKGCjt8rW9b4cY4GjqXMiX1FgjEb2uLj
9CcdjsTBErR0zr8RojOqVbFhtddxceoucWoaOJYIkiRgosSL7k6Rbbz747HYmcZSzAzHUI
SA5XSSdtVzbf74lqU0Wj0HSq6REo2JuN729vvjsdgxIlcV0oZ+VWLTpqIC2ve219/
O2Ox2OxNSJ//9k=
```


```
print(encode[:100]) #frist 100
```


```
/9j/4AAQSkZJRgABAQAAAQABAAD/
2wBDAAgFBgcGBQgHBgcJCAgJDBMMDAsLDBgREg4THBgdHRsYGxofIywlHyEqIRobJjQnKi
4v
```


Meaning of "rb"

r → Read mode (Opens the file for reading) b → Binary mode (Reads the file as binary data, not text) □ Why use "rb"?

To read non-text files (images, videos, PDFs, audio, etc.). Prevents text encoding errors that may occur if you try to read binary files in normal "r" mode.

What Does `"rb"` Mean in Python?

In Python, `"rb"` is a mode used when **reading binary data** from a file.

** Meaning of `"rb"`**

- **`r`** → **Read mode** (Opens the file for reading)
- **`b`** → **Binary mode** (Reads the file as binary data, not text)

□ **Why use `"rb"`?**

- To read **non-text files** (images, videos, PDFs, audio, etc.).
- Prevents **text encoding errors** that may occur if you try to read binary files in normal `"r"` mode.

** Example: Reading an Image File in `"rb"` Mode**

```
with open("image.jpg", "rb") as image_file:
    binary_data = image_file.read()

print(binary_data[:20])  # Print first 20 bytes
```

Why `"rb"`?

- If you use `"r"`, Python will try to decode it as text and may throw an error.

** Summary of `"rb"` vs `"r"`**

Mode	Meaning	Use Case
`"r"`	Read **text** mode	Reading text files (e.g., `.txt`, `.csv`)
`"rb"`	Read **binary** mode	Reading images, videos, PDFs, audio files

Decode Base64 Back to an Image

```
import base64
Original_image=base64.b64decode(encode) #simple

#save as a image file

with open("out_put.jpg","wb") as image_file: ##
    #
    image_file.write(Original_image) #******write()

#check your file dirctory................
```

- [ ] OIP.jpg
- [ ] out_put.jpg

Meaning of "wb"

w → Write mode (Creates a new file or overwrites an existing one)

b → Binary mode (Handles non-text files like images, videos, or any binary data)

Why use "wb"?

Because binary data (e.g., images, videos, PDFs, etc.) must be written in binary mode (b), not text mode (w).

Mode Meaning

"w" Write text mode (creates/overwrites a file)

"wb" Write binary mode (for images, videos, etc.)

"r" Read text mode

"rb" Read binary mode

Video

Just like images, videos can be encoded in Base64, but there are important things to consider.

1 How Base64 Video Encoding Works

2.Read the video file as binary data.

3.Convert the binary data to a Base64 string.

4.Use the Base64 string in HTML, JSON, or APIs.

```
import base64

with open("snowfall-in-forest.3840x2160.mp4","rb") as Video_file:
    encode=base64.b64encode(Video_file.read())
    print(encode.decode())
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.
```

```
Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

solution

video----> text file encode

text------>video decode

```
with open("snowfall-in-forest.3840x2160.mp4","rb") as Video_file:
    encode=base64.b64encode(Video_file.read()).decode("utf-8")

with open("video_text.txt","w") as video_data:
    video_data.write(encode) #large data or high level video only

print("video data save to video_text.txt")
```

```
video data save to video_text.txt
```

```
print(encode[:100])
```

```
AAAAGGZ0eXBtcDQyAAAAAG1wNDJtcDQxAAAzRG1vb3YAAABsbXZoZAAAAADgTx6s4E8erA
ABX5AAKUAAAAEAAAEAAAAAAAAAAAAA
```

decode the video

Direct method

```
decode=base64.b64decode(encode)

with open("output.mp4","wb") as Video_file:
    Video_file.write(decode)

    ##chech Your file Dirctory
```

file method

```
#use video.txt file
with open("video_text.txt","r") as video_txt:
    read_video=video_txt.read()

#decode
```

```
decode1=base64.b64decode(read_video)

#save to decoded video
with open("video_file.mp4","wb") as video_file:
    video_file.write(decode)

print("Video successfully saved as video_file.mp4")

Video successfully saved as video_file.mp4
```

importent Note

ASCII Issues & Limitations

ASCII (American Standard Code for Information Interchange) is a **7-bit character encoding system** that represents **128 characters** (0-127).

While ASCII is simple and widely used, it has many limitations, especially in modern applications.

---

1. Major Issues with ASCII

1 Limited Character Set (Only 128 Characters)

- ASCII only supports **English letters (A-Z, a-z), numbers (0-9), and basic symbols**.
- It **does NOT support non-English characters** (e.g., `é, ü, ñ, नमस्ते, 你好, 😊`).

  Solution:** Use **UTF-8**, which supports all languages.

---

2 No Support for Emojis & Special Symbols

- ASCII does **not include emojis, mathematical symbols, or currency symbols** like ₹, €, ¥.
- Many modern applications (chat, social media, AI) require support for **emojis and rich text**.
- **Solution:** Use **UTF-8**, which supports emojis (😊 in UTF-8: `\xf0\x9f\x98\x8a`).

---

3 Compatibility Issues with Non-English Languages

- ASCII **cannot encode** characters from **French, Spanish, Arabic, Hindi, Chinese, Japanese, etc.**.
- Example:

```
text = "Bonjour! Ça va?"
ascii_bytes = text.encode("ascii")  # This will throw an error
```

Error:

```
UnicodeEncodeError: 'ascii' codec can't encode character '\xe7'
```

- **Solution:** Convert text to **UTF-8** before encoding:

```
utf8_bytes = text.encode("utf-8")  # □ No error
```

---

4 ASCII is Not Space-Efficient

- ASCII uses **1 byte per character**, but it **wastes space for English text**.
- **UTF-8** is better because it uses **variable-length encoding (1-4 bytes per character)**.

---

5 No Built-in Support for File Encoding Detection

- Many old systems assume ASCII, leading to **misinterpretation of UTF-8/UTF-16 files**.
- Example: If a file contains **UTF-8 text**, but the system assumes **ASCII**, it may display `????` or ▒▒▒▒.

Solution: Always **explicitly specify UTF-8 encoding** when reading/writing files:

```
with open("file.txt", "w", encoding="utf-8") as f:
    f.write("こんにちは (Hello in Japanese)")
```

□ 2. Why Move from ASCII to UTF-8?

Feature	**ASCII** (7-bit)	**UTF-8** (Variable-length)
Character Limit	128 (A-Z, a-z, 0-9, symbols)	1.1 million+
Multilingual Support	□ No	□ Yes (All languages)
Emoji Support	□ No	□ Yes
Space Efficiency	□ Wastes space	□ Optimized
API & Web Compatibility	□ Old systems	□ Modern standard

□ 3. ASCII & UTF-8 in Base64 Encoding

□ **Base64 converts binary data into text** but does NOT handle character encoding.

- If you encode **ASCII text** in Base64, it works fine.
- If you encode **UTF-8 text**, you must convert it to bytes first.

□ UTF (Unicode Transformation Format) – Full Explanation

UTF (Unicode Transformation Format) is a set of encoding standards used to represent characters from all languages, symbols, and emojis in a universal format.

□ UTF is part of the Unicode standard, which assigns a unique number (code point) to every character in every language.

□ 1. Why Do We Need UTF Encoding?

Before UTF, different languages used different encoding systems (ASCII, ISO-8859, Shift-JIS, etc.), causing compatibility issues.

UTF solves this by providing a universal way to encode text, ensuring consistency across platforms.

Encoding	Characters Supported	Size per Character
ASCII (7-bit)	English letters (A-Z, a-z, 0-9, symbols)	1 byte (7-bit)
ISO-8859-1 (Latin-1)	Western European characters	1 byte
UTF-8	All languages, emojis, special symbols	1-4 bytes
UTF-16	All languages, optimized for Asian scripts	2 or 4 bytes
UTF-32	All languages, fixed size	4 bytes

□ 2. Different UTF Formats

a) UTF-8 (Most Common & Efficient)

- **Variable length:** 1 to 4 bytes per character.
- **Compatible with ASCII:** If a file contains only ASCII characters, it remains unchanged.
- **Efficient for English & Western languages (1 byte per character).**
- **Used in Web, APIs, JSON, XML, Python, Linux, and modern apps.**

□ Example (UTF-8 Encoding in Python)

```
text = "Hello, 😊 தமிழ் மொழி"
utf8_bytes = text.encode("utf-8")
print(utf8_bytes)  # Output: b'Hello, \xf0\x9f\x98\x8a \xe0\xa4\
xa8...'
```

□ UTF-8 Decoding

```
decoded_text = utf8_bytes.decode("utf-8")
print(decoded_text)  # Output: Hello, 😊
```

 Best Choice for APIs, web, and storage.

---

b) UTF-16 (Good for Asian Languages)

- **Fixed size of 2 bytes (or 4 bytes for rare characters).**
- **Better for languages like Chinese, Japanese, Korean (CJK), where many characters need 2 bytes.**
- **Used in Microsoft Windows, Java, and some databases.**

 Example (UTF-16 Encoding)

```
utf16_bytes = text.encode("utf-16")
print(utf16_bytes)  # Output: b'\xff\xfeH\x00e\x00l\x00l\x00o\x00,...'
```

 UTF-16 Decoding

```
decoded_text = utf16_bytes.decode("utf-16")
print(decoded_text)  # Output: Hello, 😊
```

 Not ideal for web use (wastes space for English text).

---

c) UTF-32 (Simpler but Uses More Space)

- **Fixed size: 4 bytes per character (even for simple English text).**
- **Good for processing characters directly (easier indexing).**
- **Used in some internal text processing systems.**

 Example (UTF-32 Encoding)

```
utf32_bytes = text.encode("utf-32")
print(utf32_bytes)  # Output: b'\xff\xfe\x00\x00H\x00\x00\x00e\x00\
x00...'
```

 UTF-32 Decoding

```
decoded_text = utf32_bytes.decode("utf-32")
print(decoded_text)  # Output: Hello, 😊
```

 Not space-efficient, mainly used for specialized applications.

---

3. UTF in APIs and Data Transfer (Base64 & JSON Example)

- Most APIs and databases use **UTF-8** to store and transmit text.
- **Base64 encoding is used for binary data** (e.g., images, videos) but still stores **text in UTF-8**.

Example: Sending UTF-8 Text via API (Base64 Encoded)

```
import base64
import requests

text = "Hello, 😊 தமிழ் மொழி"
utf8_bytes = text.encode("utf-8")
base64_text = base64.b64encode(utf8_bytes).decode()

data = {
    "message": base64_text  # Sending Base64-encoded UTF-8 text
}

response = requests.post("https://api.example.com/process", json=data)
```

Ensures all characters, including emojis & multilingual text, are preserved.

Summary: Which UTF Format to Use?

UTF Type	Pros	Cons	Best Use Case
UTF-8	Compact, ASCII-compatible, efficient	Variable-length	Web, APIs, JSON, modern apps
UTF-16	Good for Asian scripts (CJK)	Wastes space for English text	Windows, Java, databases
UTF-32	Fixed-size, easy indexing	Wastes too much space	Specialized processing

UTF-8 is the best choice for most applications!

```
#example
import base64

text = "தமிழ்"  # ASCII text
ascii_bytes = text.encode("ascii")  # Convert to ASCII bytes
base64_encoded = base64.b64encode(ascii_bytes).decode()  # Convert to
Base64

print(base64_encoded)  # Output: SGVsbG8=
```

```
---------------------------------------------------------------------------
UnicodeEncodeError                        Traceback (most recent call last)
Cell In[1], line 4
      1 import base64
      3 text = "தமிழ்"  # ASCII text
----> 4 ascii_bytes = text.encode("ascii")  # Convert to ASCII bytes
      5 base64_encoded = base64.b64encode(ascii_bytes).decode()  # Convert to Base64
      7 print(base64_encoded)

UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-4: ordinal not in range(128)
```

```python
import base64
text="தமிழ்" ##UFT
encode=base64.b64encode(text.encode())
print(encode)
print(encode.decode("utf-8")) #Binary to string
decode=base64.b64decode(encode)
print(decode)
print(decode.decode())
```

```
b'4K6k4K6u4K6/4K604K+N'
4K6k4K6u4K6/4K604K+N
b'\xe0\xae\xa4\xe0\xae\xae\xe0\xae\xbf\xe0\xae\xb4\xe0\xaf\x8d'
தமிழ்
```

```python
import base64
text="தமிழ்" #original text
#utf-8 bites covertion step_1

utf_code=(text.encode()) #default utf-8
print(f"step: 1  bites==frist_out>{utf_code}")


#bites to string step_2
base64_encode=base64.b64encode(utf_code)
print(f"step:2  bites to string{base64_encode}")


#decode process
base64_decode=base64.b64decode(base64_encode)
print(f"step:3 reverse the process string to bites step 3{base64_decode}")


#utf-8 decoding
utf_decoding=base64_decode.decode()
```

```
print(f"final output {utf_decoding}")


step: 1  bites==frist_out>b'\xe0\xae\xa4\xe0\xae\xae\xe0\xae\xbf\xe0\
xae\xb4\xe0\xaf\x8d'
step:2  bites to stringb'4K6k4K6u4K6/4K604K+N'
step:3 reverse the process string to bites step 3b'\xe0\xae\xa4\xe0\
xae\xae\xe0\xae\xbf\xe0\xae\xb4\xe0\xaf\x8d'
final output தமிழ்

#emoji
emoji="□"
encode=base64.b64encode(emoji.encode()).decode()

decode=base64.b64decode(encode)
print(decode.decode())

□
```

□ Workflow Explanation: Encoding & Decoding Process

You provided different representations of the Tamil word **"தமிழ்"** using **UTF-8 and Base64 encoding**. Let's break down the entire process step by step.

□ Step 1: Original Text

The Tamil word:

```
தமிழ்
```

□ Step 2: UTF-8 Encoding

When you encode **"தமிழ்"** in UTF-8, it gets converted into a byte sequence:

```
text = "தமிழ்"
utf8_encoded = text.encode("utf-8")
print(utf8_encoded)
```

□ Output (UTF-8 Byte Sequence):

```
b'\xe0\xae\xa4\xe0\xae\xae\xe0\xae\xbf\xe0\xae\xb4\xe0\xaf\x8d'
```

This byte sequence represents each Tamil character in **UTF-8 format**.

Tamil Letter	UTF-8 Byte Representation
க	`\xe0\xae\xa4`
ம	`\xe0\xae\xae`
ி	`\xe0\xae\xbf`
ன்	`\xe0\xae\xb4`
்	`\xe0\xaf\x8d`

 At this stage, we have a binary representation of the text.

Step 3: Base64 Encoding

Now, we **encode the UTF-8 byte sequence into Base64** to safely transmit/store it in text format (for APIs, JSON, etc.).

```
import base64

base64_encoded = base64.b64encode(utf8_encoded)
print(base64_encoded.decode())  # Convert bytes to string
```

 Output (Base64 Encoded String):

```
4K6k4K6u4K6/4K604K+N
```

 Base64 converts binary data into an ASCII-safe string format.

Step 4: Base64 Decoding (Reverse Process)

To get back the original **UTF-8 bytes**, we **decode the Base64 string**:

```
base64_decoded = base64.b64decode("4K6k4K6u4K6/4K604K+N")
print(base64_decoded)  # Output: UTF-8 bytes
```

 Output:

```
b'\xe0\xae\xa4\xe0\xae\xae\xe0\xae\xbf\xe0\xae\xb4\xe0\xaf\x8d'
```

 This matches our original UTF-8 encoded bytes!

Step 5: UTF-8 Decoding (Final Step)

Now, we **decode the UTF-8 bytes back into readable Tamil text**:

```
decoded_text = base64_decoded.decode("utf-8")
print(decoded_text)
```

 Output:

```
தமிழ்
```

 We successfully recovered the original Tamil text!

Summary of the Workflow

Step	Process	Input	Output
1	Original Text	தமிழ்	தமிழ்
2	UTF-8 Encoding	தமிழ்	`b'\xe0\xae\xa4\xe0\xae\xae\xe0\xae\xbf\xe0\xae\xb4\xe0\xaf\x8d'`
3	Base64 Encoding	`b'\xe0\xae\xa4\xe0\xae\xae\xe0\xae\xbf\xe0\xae\xb4\xe0\xaf\x8d'`	`"4K6k4K6u4K6/4K604K+N"`
4	Base64 Decoding	`"4K6k4K6u4K6/4K604K+N"`	`b'\xe0\xae\xa4\xe0\xae\xae\xe0\xae\xbf\xe0\xae\xb4\xe0\xaf\x8d'`
5	UTF-8 Decoding	`b'\xe0\xae\xa4\xe0\xae\xae\xe0\xae\xbf\xe0\xae\xb4\xe0\xaf\x8d'`	தமிழ்

'

```
<source src="decode1" type="video/mp4">
```

view the video

JSON Format for LLM Model Requests (API Integration)

If you're working with **LLMs (Large Language Models)** like **OpenAI, Gemini, Llama, Mistral, etc.**, you need to send data in **JSON format**.

Here's how you structure JSON for different LLM tasks:

---

1. Basic JSON for LLM Chat API (OpenAI Example)

This is a typical **conversation-based request**:

```
{
  "model": "gpt-4",
  "messages": [
    {"role": "system", "content": "You are a helpful AI assistant."},
    {"role": "user", "content": "What is AI?"}
  ],
  "temperature": 0.7,
  "max_tokens": 500
}
```

Breakdown:

- `"model"` → The LLM model to use (e.g., `gpt-4`, `gpt-3.5-turbo`).
- `"messages"` → The conversation history:
  - `"system"` → Defines AI behavior.
  - `"user"` → User's question or input.
- `"temperature"` → Controls randomness (`0.7` = creative, `0.2` = more deterministic).
- `"max_tokens"` → Limits response length.

API Request in Python:

```
import requests

url = "https://api.openai.com/v1/chat/completions"
headers = {"Authorization": "Bearer YOUR_API_KEY", "Content-Type":
"application/json"}
data = {
    "model": "gpt-4",
    "messages": [
        {"role": "system", "content": "You are a helpful AI
assistant."},
        {"role": "user", "content": "What is AI?"}
    ],
    "temperature": 0.7,
    "max_tokens": 500
}

response = requests.post(url, headers=headers, json=data)
print(response.json())
```

---

2. JSON for LLM Text Completion API

Some models require a **single prompt** instead of a chat history:

```
{
  "model": "text-davinci-003",
  "prompt": "Explain machine learning in simple words.",
  "temperature": 0.7,
  "max_tokens": 200
}
```

Used for → Text generation without multi-turn chat.

API Request in Python:

```
data = {
    "model": "text-davinci-003",
    "prompt": "Explain machine learning in simple words.",
    "temperature": 0.7,
    "max_tokens": 200
}

response = requests.post(url, headers=headers, json=data)
print(response.json())
```

---

3. JSON for Image + Text (Multimodal LLMs like GPT-4V, Gemini, LLaVA)

If you want to **send an image along with text**, the image needs to be **Base64 encoded**:

```
{
  "model": "gpt-4-vision-preview",
  "messages": [
    {"role": "user", "content": [
        {"type": "text", "text": "What is in this image?"},
        {"type": "image_url", "image_url":
"..."}
    ]}
  ],
  "max_tokens": 500
}
```

API Request in Python (With Image):

```
import base64
```

```python
# Read and encode image
with open("image.png", "rb") as img:
    base64_image = base64.b64encode(img.read()).decode()

# Prepare request data
data = {
    "model": "gpt-4-vision-preview",
    "messages": [
        {"role": "user", "content": [
            {"type": "text", "text": "What is in this image?"},
            {"type": "image_url", "image_url":
f"{base64_image}"}
        ]}
    ],
    "max_tokens": 500
}

# Send request
response = requests.post(url, headers=headers, json=data)
print(response.json())
```

---

4. JSON for Code Generation (Codex / GPT-4 for Coding)

To generate code in a specific language:

```json
{
  "model": "gpt-4",
  "messages": [
    {"role": "system", "content": "You are a Python expert."},
    {"role": "user", "content": "Write a Python function to reverse a
string."}
  ],
  "temperature": 0.2,
  "max_tokens": 150
}
```

 Used for → AI-powered **code generation, debugging, and explanation**.

---

5. JSON for LLM Fine-Tuning Data

If you are **fine-tuning a model**, you need to provide **training data** in JSONL format:

```
{"messages": [{"role": "system", "content": "You are an AI tutor."},
{"role": "user", "content": "Explain gravity."}, {"role": "assistant",
"content": "Gravity is the force that pulls objects toward each
other."}]}
{"messages": [{"role": "system", "content": "You are an AI tutor."},
{"role": "user", "content": "What is Newton's first law?"}, {"role":
"assistant", "content": "Newton's first law states that an object in
motion stays in motion unless acted upon by an external force."}]}
```

 Used for → Training AI on **custom data** to improve responses.

Summary Table

Use Case	JSON Example
Chat Model (GPT-4, Gemini, Llama-3)	`{ "model": "gpt-4", "messages": [...] }`
Text Completion (Davinci, Mistral, Falcon)	`{ "model": "text-davinci-003", "prompt": "...", "temperature": 0.7 }`
Image + Text (GPT-4V, Gemini Pro Vision)	`{ "model": "gpt-4-vision-preview", "messages": [...] }`
Code Generation (Codex, GPT-4 Turbo)	`{ "model": "gpt-4", "messages": [...] }`
Fine-Tuning Data	`{"messages": [{"role": "user", "content": "..."}]}`

Base64 Use Cases in Different Domains

Base64 is widely used in **web development, APIs, networking, cryptography, and data storage**. Here are some **real-world use cases**:

---

1 APIs: Sending Images, Videos, & Files

Many APIs **do not support raw binary data**. Instead, Base64 encodes **images, videos, or files** as text to send over HTTP.

 Example: Sending an image in JSON (API request)

```
import base64
import requests
```

```python
# Read and encode an image
with open("image.jpg", "rb") as img_file:
    base64_image = base64.b64encode(img_file.read()).decode()

# Send in JSON
data = {"image": base64_image}
url = "https://example.com/api/upload"
response = requests.post(url, json=data)

print(response.json())
```

 Used in: REST APIs (AI models, file uploads, cloud storage).

---

2 Embedding Images in Webpages (HTML & CSS)

Instead of hosting images separately, Base64 allows **direct embedding in HTML/CSS**.

 Example: Base64 image in HTML

```html
<img src="..." />
```

 Used in: Email templates, single-page applications, web performance optimization.

---

3 JSON Web Tokens (JWT) & Authentication

Base64 is used in **JWT tokens** for secure authentication in web apps.

 Example JWT token (Base64 Encoded)

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

 Used in: User authentication (OAuth, Firebase, API security).

---

4 Sending Attachments in Emails (MIME Encoding)

Email systems use Base64 to encode file attachments.

 Example: Email attachment (Base64 Encoded)

```
Content-Type: image/png
Content-Transfer-Encoding: base64
iVBORw0KGgoAAAANSUhEUg...
```

 Used in: Gmail, Outlook, SMTP-based email services.

---

5 Storing Images & Files in Databases

Databases **don't store images natively**, so Base64 helps store them as text.

 Example: Store & retrieve images in a database

```
import sqlite3, base64

# Store image
image_data = base64.b64encode(open("image.png", "rb").read()).decode()
conn = sqlite3.connect("mydb.db")
cursor = conn.cursor()
cursor.execute("INSERT INTO images (name, data) VALUES (?, ?)",
("profile_pic", image_data))
conn.commit()
```

 Used in: MongoDB, MySQL, PostgreSQL, Firebase.

---

6 Encoding Binary Data in URLs

Base64 helps encode **binary files** in URLs without breaking them.

 Example: Encoding text for a GET request

```
import urllib.parse
import base64

text = "Hello, Vicky!"
encoded = base64.b64encode(text.encode()).decode()
safe_url = urllib.parse.quote(encoded)

print(safe_url)
```

 Used in: Web APIs, QR codes, URL shorteners.

---

7 Encryption & Data Security

Base64 helps encode **keys, credentials, and sensitive data** before transmission.

 Example: Encoding API keys

```
api_key = "my-secret-key"
encoded_key = base64.b64encode(api_key.encode()).decode()
```

 Used in: Cryptography, secure key storage.

Summary of Base64 Use Cases

Use Case	Where Used?
APIs (Images, Videos, Files)	AI models, cloud storage, REST APIs
Embedding Images in Web	HTML, CSS, emails
Authentication (JWT Tokens)	OAuth, Firebase, API security
Email Attachments (MIME Encoding)	Gmail, SMTP, Outlook
Storing Images in Databases	MongoDB, MySQL, Firebase
Encoding Data in URLs	Web APIs, QR codes
Encryption & Data Security	Cryptography, password storage

LLM (Large Language Models) & Base64 Use Cases

Base64 is often used in **LLM-based APIs** for processing **images, videos, documents, or other binary data**.

---

1. Why Use Base64 in LLM Models?

 LLM APIs (like OpenAI, Google Gemini, LLaMA, Claude, Mistral) use JSON-based APIs that do not support raw binary data.
 Base64 allows encoding images, PDFs, and videos into a text format that can be sent in an API request.
 Ensures **safe transmission** over HTTP without breaking the request format.

---

2. Sending an Image to an LLM (OpenAI GPT-4V Example)

Let's send a **Base64-encoded image** to OpenAI's **GPT-4 Vision API** to analyze its content.

 Step 1: Convert Image to Base64

```
import base64

with open("image.jpg", "rb") as img:
    base64_image = base64.b64encode(img.read()).decode()

print(base64_image[:100])  # Preview first 100 characters
```

 Step 2: Send Image to OpenAI API

```
import requests

url = "https://api.openai.com/v1/chat/completions"
headers = {
    "Authorization": "Bearer YOUR_OPENAI_API_KEY",
    "Content-Type": "application/json"
}

# API Request Data
data = {
    "model": "gpt-4-vision-preview",
    "messages": [
        {"role": "user", "content": [
            {"type": "text", "text": "Describe this image."},
            {"type": "image_url", "image_url":
f"{base64_image}"}
        ]}
    ],
    "max_tokens": 500
}

response = requests.post(url, headers=headers, json=data)
print(response.json())
```

 Use Case: Image-to-text generation, object recognition, AI-powered analysis.

---

3. Sending a PDF Document to LLM (Claude AI Example)

Some LLMs can process **PDF documents** by converting them into Base64.

 Example: Encode & Send a PDF to Claude AI

```
import base64
import requests

# Convert PDF to Base64
with open("document.pdf", "rb") as pdf:
    base64_pdf = base64.b64encode(pdf.read()).decode()
```

```
# API Request Data
data = {
    "model": "claude-3",
    "messages": [
        {"role": "user", "content": [
            {"type": "text", "text": "Summarize this document."},
            {"type": "file", "file": f"data:application/pdf; base64,
{base64_pdf}"}
        ]}
    ]
}

url = "https://api.anthropic.com/v1/messages"
headers = {
    "x-api-key": "YOUR_ANTHROPIC_API_KEY",
    "Content-Type": "application/json"
}

response = requests.post(url, headers=headers, json=data)
print(response.json())
```

 Use Case: Document summarization, legal analysis, research paper review.

---

4. Sending a Video Frame to an LLM (Google Gemini Example)

Some LLMs (like **Google Gemini** and OpenAI's **GPT-4V**) can analyze **video frames** by encoding images as Base64.

 Extracting & Encoding a Video Frame

```
import cv2
import base64

# Read a frame from a video
video = cv2.VideoCapture("video.mp4")
ret, frame = video.read()

if ret:
    _, buffer = cv2.imencode(".jpg", frame)  # Convert frame to JPG
    base64_frame = base64.b64encode(buffer).decode()

print(base64_frame[:100])  # Preview first 100 characters
```

 Use Case: AI-based video analysis, action recognition, scene understanding.

---

5. Base64 in LLM Chatbots & APIs

Example: Sending a Base64-encoded document to a chatbot

```
{
    "model": "llama-3",
    "messages": [
        {"role": "user", "content": [
            {"type": "text", "text": "Extract key insights from this
file."},
            {"type": "file", "file":
"data:application/pdf; base64,JVBERi0xLjQKJc..."}
        ]}
    ]
}
```

Use Case: LLM-powered file processing, research automation.

Summary of LLM & Base64 Use Cases

LLM Use Case	Base64 Required?	Example API
Image Processing	Yes	OpenAI GPT-4V, Gemini, Claude
PDF Analysis	Yes	Claude, GPT-4 Turbo
Video Frame Analysis	Yes	GPT-4V, Gemini
Text-to-Speech (TTS) Audio Files	Yes	OpenAI Whisper, Azure Speech
Chatbots with File Inputs	Yes	LLaMA, Mistral, GPT