In [192…
```
import base64
import json
import hmac
import hashlib
import time
```

In [193…
```
type('abc')
type(b'abc')
```

Out[193… str

Out[193… bytes

In [194…
```
data = b'abc'
len(data)
data[0]  # each byte is a number 0-255
data[1]
data[2]
```

Out[194… 3

Out[194… 97

Out[194… 98

Out[194… 99

In [195…
```
type(data[0])  # bytes is like [int, int, ...]

chr(data[0])  # integer to ascii
chr(data[1])
chr(data[2])
```

Out[195… int

Out[195… 'a'

Out[195… 'b'

Out[195… 'c'

In [196…
```
chr(97)
chr(98)
chr(99)
```

Out[196… 'a'

Out[196… 'b'

Out[196… 'c'

In [197…
```
chr(0)  # 0-31 are control chars
chr(1)
chr(2)
chr(15)
```

```
chr(16)
chr(255)  # last char of 8 bit ascii
```

Out[197… '\x00'

Out[197… '\x01'

Out[197… '\x02'

Out[197… '\x0f'

Out[197… '\x10'

Out[197… 'ÿ'

\xHH is hexadecimal byte escape

HH = exactly 2 hex digits

Represents one byte (0–255)

In [198…
```
'\x41'  # hex 41 = dec 65
'\x42'
'\x43'
```

Out[198… 'A'

Out[198… 'B'

Out[198… 'C'

In [199…
```
type('\x41')
'\x41' == 'A'
```

Out[199… str

Out[199… True

In [200…
```
print('x41 is my best friend')
print('\x41 is my best friend')
print('\\x41 is my best friend')
```

```
x41 is my best friend
A is my best friend
\x41 is my best friend
```

In [201…
```
print('apple/orange')  # forward slash in ok
```

```
apple/orange
```

In [202…
```
print('apple\\orange')  # \\ -> \
print(r'apple\orange')  # raw string
```

```
apple\orange
apple\orange
```

In [203…
```
data  # bytes
data.decode()  # str
```

Out[203… b'abc'

Out[203… 'abc'

In [204…
```
f'{97}'
f'{97:.2f}'  # format specification after `:`

f'{97551}'
f'{97551:,}'
f'{97551:,.2f}'
```

Out[204… '97'

Out[204… '97.00'

Out[204… '97551'

Out[204… '97,551'

Out[204… '97,551.00'

In [205…
```
f'{0:b}'  # binary representation of integers
f'{1:b}'
f'{2:b}'
f'{3:b}'
f'{4:b}'
f'{18:b}'
```

Out[205… '0'

Out[205… '1'

Out[205… '10'

Out[205… '11'

Out[205… '100'

Out[205… '10010'

In [206…
```
f'{0:b}'  # exact binary
f'{0:5b}'  # min len = 5
f'{0:05b}'  # padding with 0
f'{18:05b}'  # no padding needed
```

Out[206… '0'

Out[206… '   0'

Out[206… '00000'

Out[206… '10010'

In [207…
```
for i in data:
  print(i, chr(i), f'{i:08b}')
```

```
97 a 01100001
98 b 01100010
99 c 01100011
```

In [208…
```
list(b'abc')  # bytes to decimal

list(b'vidu is my hero')
```

Out[208…
```
[97, 98, 99]
```

Out[208…
```
[118, 105, 100, 117, 32, 105, 115, 32, 109, 121, 32, 104, 101, 114, 111]
```

In [209…
```
bytes([97, 98, 99])  # decimal to bytes

bytes([
    118, 105, 100, 117, 32, 105, 115, 32, 109, 121, 32, 104, 101, 114, 111
])
```

Out[209…
```
b'abc'
```

Out[209…
```
b'vidu is my hero'
```

In [210…
```
' '.join(
    f'{i:08b}' for i in [
        118, 105, 100, 117, 32, 105, 115, 32, 109, 121, 32, 104, 101, 114, 111
    ]
)  # full binary sequence for `vidu is my hero`
```

Out[210…
```
'01110110 01101001 01100100 01110101 00100000 01101001 01110011 00100000 01101101
01111001 00100000 01101000 01100101 01110010 01101111'
```

In [211…
```
int('01110110', 2)  # binary to int
chr(int('01110110', 2))  # int to char
```

Out[211…
```
118
```

Out[211…
```
'v'
```

In [212…
```
binary_str = '01110110 01101001 01100100 01110101 00100000 01101001 01110011 001000

for byte_str in binary_str.split(' '):
  print(byte_str, int(byte_str, 2), chr(int(byte_str, 2)))
```

```
01110110 118 v
01101001 105 i
01100100 100 d
01110101 117 u
00100000 32
01101001 105 i
01110011 115 s
00100000 32
01101101 109 m
01111001 121 y
00100000 32
01101000 104 h
01100101 101 e
01110010 114 r
01101111 111 o
```

implementing base64 encoding/decoding from scratch

goal of base64

encoding: bytes -> str

decoding: str -> bytes

only 64 symbols (`base64_symbols`) are allowed in encoded str

In [213…
```python
import string
base64_symbols = string.ascii_uppercase + string.ascii_lowercase + string.digits +
base64_symbols
len(base64_symbols)  # 26 + 26 + 10 + 2
```

Out[213…
```
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
```

Out[213…
```
64
```

encoding: mapping nums to symbols

0–25 → A–Z

26–51 → a–z

52–61 → 0–9

62 → +

63 → /

In [214…
```python
for i, symbol in enumerate(base64_symbols):
  print(i, f'{i:06b}', symbol)  # 6 bits: 2^6 = 64 states (0-63 in decimal)
```

0 000000 A
1 000001 B
2 000010 C
3 000011 D
4 000100 E
5 000101 F
6 000110 G
7 000111 H
8 001000 I
9 001001 J
10 001010 K
11 001011 L
12 001100 M
13 001101 N
14 001110 O
15 001111 P
16 010000 Q
17 010001 R
18 010010 S
19 010011 T
20 010100 U
21 010101 V
22 010110 W
23 010111 X
24 011000 Y
25 011001 Z
26 011010 a
27 011011 b
28 011100 c
29 011101 d
30 011110 e
31 011111 f
32 100000 g
33 100001 h
34 100010 i
35 100011 j
36 100100 k
37 100101 l
38 100110 m
39 100111 n
40 101000 o
41 101001 p
42 101010 q
43 101011 r
44 101100 s
45 101101 t
46 101110 u
47 101111 v
48 110000 w
49 110001 x
50 110010 y
51 110011 z
52 110100 0
53 110101 1
54 110110 2
55 110111 3

```
56 111000 4
57 111001 5
58 111010 6
59 111011 7
60 111100 8
61 111101 9
62 111110 +
63 111111 /
```

In [215…
```python
data = b'vidu is my hero'
type(data)
```

Out[215…
```
bytes
```

step 1: bytes -> binary str

In [216…
```python
binary_str = ''.join(f'{byte:08b}' for byte in data)
binary_str
len(binary_str)
len(binary_str) % 6
```

Out[216…
```
'011101100110100101100100011101010010000001101001011100110010000001101101011110010
010000001101000011001010111001001101111'
```

Out[216…
```
120
```

Out[216…
```
0
```

step 2: split binary str in chunks of 6 bits

In [217…
```python
# problem: split a str in parts of length x

fruits = 'applemangolemon'  # 3 words of 5 letters each

split_indices = list(range(0, len(fruits), 5))  # start, end, step
split_indices

for i in split_indices:
  print(i, i+5, fruits[i:i+5])
```

Out[217…
```
[0, 5, 10]
```

```
0 5 apple
5 10 mango
10 15 lemon
```

In [218…
```python
chunks = [binary_str[i:i+6] for i in range(0, len(binary_str), 6)]
print(chunks)
```

```
['011101', '100110', '100101', '100100', '011101', '010010', '000001', '101001', '01
1100', '110010', '000001', '101101', '011110', '010010', '000001', '101000', '01100
1', '010111', '001001', '101111']
```

step 3: chunks -> nums [0-63]

In [219…
```
nums = [int(chunk, 2) for chunk in chunks]
nums
```

Out[219…
```
[29, 38, 37, 36, 29, 18, 1, 41, 28, 50, 1, 45, 30, 18, 1, 40, 25, 23, 9, 47]
```

step 4: nums -> symbols

In [220…
```
encoded_str = ''.join([base64_symbols[num] for num in nums])
encoded_str
```

Out[220…
```
'dmlkdSBpcyBteSBoZXJv'
```

In [221…
```
base64.b64encode(data)  # python implementation
```

Out[221…
```
b'dmlkdSBpcyBteSBoZXJv'
```

decoding, step 1: symbols -> nums

In [222…
```
nums = [base64_symbols.index(symbol) for symbol in encoded_str]
nums
```

Out[222…
```
[29, 38, 37, 36, 29, 18, 1, 41, 28, 50, 1, 45, 30, 18, 1, 40, 25, 23, 9, 47]
```

step 2: nums -> binary chunks of 6 bits

In [223…
```
chunks = [f'{num:06b}' for num in nums]
print(chunks)
```

```
['011101', '100110', '100101', '100100', '011101', '010010', '000001', '101001', '011100', '110010', '000001', '101101', '011110', '010010', '000001', '101000', '011001', '010111', '001001', '101111']
```

step 3: join those chunks to get full binary str

In [224…
```
binary_str = ''.join(chunks)
binary_str
```

Out[224…
```
'011101100110100101100100011101010010000001101001011100110010000001101101011110010010000001101000011001010111001001101111'
```

step 4: binary str -> bytes

each byte = 8 bits

full process:

1. split in chunks of 8 bits
2. 8 bit chunk -> decimal num [0-255]
3. step 2 is effectively: chunks -> [int, int, ...]

4. finally: bytes([int, int, ...])

enjoy :)

In [229…
```
data = bytes(int(binary_str[i:i+8], 2) for i in range(0, len(binary_str), 8))
data
```

Out[229…
```
b'vidu is my hero'
```

In [230…
```
base64.b64decode(encoded_str)  # python implementation
```

Out[230…
```
b'vidu is my hero'
```

In [226…
```
# base64.urlsafe_b64encode(b'a')  # 8 bits data (6, 2) -> 2
# base64.urlsafe_b64encode(b'ab')  # 16 bits data (6, 6, 4) -> 3
# base64.urlsafe_b64encode(b'abc')  # 24 bits data (6, 6, 6, 6) -> 4
# base64.urlsafe_b64encode(b'abcd')  # 32 bits data (6, 6, 6, 6, 6, 2) -> 6
```