String vs. Bytes

Text in Python 3 is always Unicode and is represented by the **str** type, and binary data is represented by the **bytes** type. They cannot be mixed.

Strings can be **encoded** to bytes, and bytes can be **decoded** back to strings.

In [1]:
```
s = 'Hello world!'
print(s)
print("length is", len(s))
```

```
Hello world!
length is 12
```

In [2]:
```
us = 'Hello 世界!'
print(us)
print("length is", len(us))
```

```
Hello 世界!
length is 9
```

Now encode both strings to bytes.

In [3]:
```
bs = s.encode('utf-8')
print(bs)
print("length is", len(bs))
```

```
b'Hello world!'
length is 12
```

In [4]:
```
bus = us.encode('utf-8')
print(bus)
print("length is", len(bus))
```

```
b'Hello \xe4\xb8\x96\xe7\x95\x8c!'
length is 13
```

Decode back to strings.

In [5]:
```
print(bs.decode('utf-8'))
print(bus.decode('utf-8'))
```

```
Hello world!
Hello 世界!
```

Big Endian vs Little Endian

In [6]:
```
num = 258
print(num.to_bytes(2, "big"))
print(num.to_bytes(2, "little"))

print(num.to_bytes(4, "big"))
print(num.to_bytes(4, "little"))
```

```
b'\x01\x02'
b'\x02\x01'
b'\x00\x00\x01\x02'
b'\x02\x01\x00\x00'
```

struct package

This module performs conversions between Python values and C structs represented as Python bytes objects.

In [7]:
```
import struct
```

struct.pack(fmt, v1, v2, …)

Return a bytes object containing the values v1, v2, … packed according to the format string fmt. The arguments must match the values required by the format exactly.

1. **"!"** means network endianess (big endian)
2. **"<"** means little endian
3. **">"** means big endian
4. **"="** means native
5. **"h"** measn short integer (2 bytes)

In [8]:
```
x = 256

print("Network endianess")
print(struct.pack('!h', x))

print("Little endian")
print(struct.pack('<h', x))

print("Big endian")
print(struct.pack('>h', x))

print("Native endianess")
print(struct.pack('=h', x))
```

```
Network endianess
b'\x01\x00'
Little endian
b'\x00\x01'
Big endian
b'\x01\x00'
Native endianess
b'\x00\x01'
```

struct.unpack(fmt, buffer)

Unpack from the buffer buffer (presumably packed by pack(fmt, ...)) according to the format string fmt. The result is a tuple even if it contains exactly one item. The buffer's size in bytes must match the size required by the format,

In [9]:
```
bx = struct.pack('!h', x)

print(struct.unpack('!h', bx))
print(struct.unpack('<h', bx))
```

```
(256,)
(1,)
```

In [10]:
```
print(struct.unpack('!h', bx)[0])
print(struct.unpack('<h', bx)[0])
```

```
256
1
```