Strings vs. Bytes

Bytes are bytes; characters are an abstraction. An immutable sequence of Unicode characters is called a *string*. An immutable sequence of numbers-between-0-and-255 is called a *bytes* object.

```
by = b'abcd\x65'
print (by)
#b'abcde'

print (type(by) )          #②
#<class 'bytes'>

print (len(by) )           #③
#5

by += b'\xff'              #④
print (by)
#b'abcde\xff'

print (len(by))            #⑤
#6

print (by[0])              #⑥
#97
```

```
by = b'abcd\x65'
print (by[0] = 102)        #⑦
# File "/usercode/__ed_file.py", line 2
# print (by[0] = 102) #\u2466
# ^
#SyntaxError: keyword can't be an expression
```

① To define a `bytes` object, use the `b''` “byte literal” syntax. Each byte within the byte literal can be an `ASCII` character or an encoded hexadecimal number from `\x00` to `\xff` (0–255).

② The type of a `bytes` object is `bytes`.

③ Just like lists and strings, you can get the length of a `bytes` object with the built-in `len()` function.

④ Just like lists and strings, you can use the `+` operator to concatenate `bytes` objects. The result is a new `bytes` object.

⑤ Concatenating a 5-byte `bytes` object and a 1-byte `bytes` object gives you a 6-byte `bytes` object.

⑥ Just like lists and strings, you can use index notation to get individual bytes in a `bytes` object. The items of a string are strings; the items of a `bytes` object are integers. Specifically, integers between 0–255.

⑦ A `bytes` object is immutable; you can not assign individual bytes. If you need to change individual bytes, you can either use string slicing and concatenation operators (which work the same as strings), or you can convert the `bytes` object into a `bytearray` object.

```
by = b'abcd\x65'
barr = bytearray(by)  #①
print (barr)
#bytearray(b'abcde')

print (len(barr))     #②
#5

barr[0] = 102         #③
print (barr)
#bytearray(b'fbcde')
```

① To convert a `bytes` object into a mutable `bytearray` object, use the built-in `bytearray()` function.

② All the methods and operations you can do on a `bytes` object, you can do on a `bytearray` object too.

③ The one difference is that, with the `bytearray` object, you can assign individual bytes using index notation. The assigned value must be an integer between 0–255.

The one thing you *can never do* is mix bytes and strings.

```
by = b'd'
s = 'abcde'
print (by + s  )                    #①
#Traceback (most recent call last):
# File "/usercode/__ed_file.py", line 3, in <module>
# print (by + s ) #\u2460
#TypeError: can't concat bytes to str
```

```
by = b'd'
s = 'abcde'

print (s.count(by))                 #②
#Traceback (most recent call last):
# File "/usercode/__ed_file.py", line 4, in <module>
# print (s.count(by)) #\u2461
#TypeError: Can't convert 'bytes' object to str implicitly
```

```
by = b'd'
s = 'abcde'

print (s.count(by.decode('ascii')))  #③
#1
```

① You can’t concatenate bytes and strings. They are two different data types.

② You can’t count the occurrences of bytes in a string, because there are no bytes in a string. A string is a sequence of characters. Perhaps you meant “count the occurrences of the string that you would get after decoding this sequence of bytes in a particular character encoding”? Well then, you’ll need to say that explicitly. Python 3 won’t implicitly convert bytes to strings or strings to bytes.

③ By an amazing coincidence, this line of code says “count the occurrences of the string that you would get after decoding this sequence of bytes in this particular character encoding.”

And here is the link between strings and bytes: bytes objects have a `decode()` method that takes a character encoding and returns a string, and strings have an `encode()` method that takes a character encoding and returns a `bytes` object. In the previous example, the decoding was relatively straightforward — converting a sequence of bytes in the `ASCII` encoding into a string of characters. But the same process works with any encoding that supports the characters of the string — even legacy (non-Unicode) encodings.

```
a_string = '深入 Python'         #①

print (len(a_string))
#9

by = a_string.encode('utf-8')    #②
print (by)
#b'\xe6\xb7\xb1\xe5\x85\xa5 Python'

print (len(by))
#13

by = a_string.encode('gb18030')  #③
print (by)
#b'\xc9\xee\xc8\xeb Python'

print (len(by))
#11

by = a_string.encode('big5')     #④
print (by)
#b'\xb2`\xa4J Python'

print (len(by))
#11

roundtrip = by.decode('big5')    #⑤
roundtrip
#'深入 Python'

print (a_string == roundtrip)
#True
```

① This is a string. It has nine characters.

② This is a `bytes` object. It has 13 bytes. It is the sequence of bytes you get when you take `a_string` and encode it in `UTF-8`.

③ This is a `bytes` object. It has 11 bytes. It is the sequence of bytes you get when you take `a_string` and encode it in GB18030.

④ This is a `bytes` object. It has 11 bytes. It is an entirely *different sequence of bytes* that you get when you take `a_string` and encode it in Big5.

⑤ This is a string. It has nine characters. It is the sequence of characters you get when you take `by` and decode it using the Big5 encoding algorithm. It is identical to the original string.