Byte. In a science fiction movie, the great truth of reality is revealed to you. The universe is composed of units (indivisible units) like atoms (or bytes).

With bytes, we have an addressable unit of memory. A byte is made of bits—but these are not as easily accessed. A byte can store 0 through 255—we use bytes and bytearray.

Bytearray example. This example creates a list. Each number in the list is between 0 and 255 (inclusive). We create a bytearray from the list.

Modify:
We modify the first 2 elements in the bytearray. This cannot be done with a bytes object.

For:
We use the for-loop to iterate over the bytearray's elements. This is the same as how we use a list.

For

```
Based on:     Python 3 (2018)

Python program that creates bytearray from list
```

```
elements = [0, 200, 50, 25, 10, 255]

# Create bytearray from list of integers.
values = bytearray(elements)

# Modify elements in the bytearray.
values[0] = 5
values[1] = 0

# Display bytes.
for value in values:
    print(value)
```

Output

```
5
0
50
25
10
255
```

Bytes example. We now consider "bytes." This is similar to bytearray. But the elements of a bytes object cannot be changed. It is an immutable array of bytes.

Buffer protocol:
Bytearray, bytes and memoryview act upon the buffer protocol. They all share similar syntax with small differences.

Python program that creates bytes object

```
elements = [5, 10, 0, 0, 100]

# Create immutable bytes object.
data = bytes(elements)
```

```
# Loop over bytes.
for d in data:
    print(d)
```

Output

```
5
10
0
0
100
```

Error. Now we get into some trouble—that is always fun. Here we try to modify the first element of a bytes object. Python complains—the "object does not support item assignment."

Python program that causes error

```
data = bytes([10, 20, 30, 40])

# We can read values from a bytes object.
print(data[0])

# We cannot assign elements.
data[0] = 1
```

Output

```
10
Traceback (most recent call last):
  File "/Users/sam/Documents/test.py", line 9, in <module>
    data[0] = 1
TypeError: 'bytes' object does not support item assignment
```

Len. We can get the length of a bytearray or bytes object with the len built-in. Here we use bytearray in the same way as a string or a list.

Len

Python program that uses len, gets byte count

```
# Create bytearray from some data.
values = bytearray([6, 7, 60, 70, 0])

# It has 5 elements.
# ... The len is 5.
print("Element count:", len(values))
```

Output

```
Element count: 5
```

Literals. Bytes and bytearray objects can be created with a special string literal syntax. We prefix the literals with a "b." This prefix is required.

Tip:

Buffer protocol methods require byte-prefix string literals, even for arguments to methods like replace().

Python program that uses byte literals

```
# Create bytes object from byte literal.
data = bytes(b"abc")
for value in data:
    print(value)
```

```
print()

# Create bytearray from byte literal.
arr = bytearray(b"abc")
for value in arr:
    print(value)
```

Output

```
97
98
99

97
98
99
```

Slice, bytearray. We can slice bytearrays. And because bytearray is mutable, we can use slices to change its contents. Here we assign a slice to an integer list.

Python program that uses slice, changes bytearray

```
values = [5, 10, 15, 20]
arr = bytearray(values)

# Assign first two elements to new list.
arr[0:2] = [100, 0, 0]

# The array is now modified.
for v in arr: print(v)
```

```
Output

100
0
0
15
20
```

Slice, bytes. A bytes object too supports slice syntax, but it is read-only. Here we get a slice of bytes (the first two elements) and loop over it.

Often:
We can loop over a slice directly in the for-loop condition. The variable is not needed.

```
Python program that uses slice, bytes

data = bytes(b"abc")

# Get a slice from the bytes object.
first_part = data[0:2]

# Display values from slice.
for element in first_part: print(element)
```

```
Output

97
98
```

Count. Many methods are available on the buffer interface. Count is one. It loops through the bytes and counts instances matching our specified pattern.

Note:

Count must loop through all elements. If another loop is needed afterwards, often we can combine loops for speed.

Argument:

The argument to count() must be a byte object, like a "b" string literal or a number between 0 and 255.

Python program that uses count, buffer interface

```
# Create a bytes object and a bytearray.
data = bytes(b"aabbcccc")
arr = bytearray(b"aabbcccc")

# The count method (from the buffer interface) works on both.
print(data.count(b"c"))
print(arr.count(b"c"))
```

Output

```
4
4
```

Find. This method returns the leftmost index of a matching sequence. Optionally we can specify a start index and an end index (as the second and third arguments).

Python program that uses find

```
data = bytes(b"python")

# This sequence is found.
```

```
index1 = data.find(b"on")
print(index1)

# This sequence is not present.
index2 = data.find(b"java")
print(index2)
```

Output

```
4
-1
```

In operator. This tests for existence. We use "in" to see if an element exists within the bytes objects. This is a clearer way to see if a byte exists in our object.

Python program that uses in operator

```
data = bytes([100, 20, 10, 200, 200])

# Test bytes object with "in" operator.
if 200 in data:
    print(True)

if 0 not in data:
    print(False)
```

Output

```
True
False
```

Combine two bytearrays. As with lists and other sequences, we can combine two bytearrays (or bytes) with a plus. In my tests, I found it does not matter if we combine two different types.

Python program that uses plus on bytearrays

```
left = bytearray(b"hello ")
right = bytearray(b"world")

# Combine two bytearray objects with plus.
both = left + right
print(both)
```

Output

```
bytearray(b'hello world')
```

Convert list. A list of bytes (numbers between 0 and 256) can be converted into a bytearray with the constructor. To convert back into a list, please use the list built-in constructor.

Tip:

Lists display in a more friendly way with the print method. So we might use this code to display bytearrays and bytes.

Python program that uses list built-in

```
initial = [100, 255, 255, 0]
print(initial)

# Convert the list to a byte array.
b = bytearray(initial)
print(b)
```

```
# Convert back to a list.
result = list(b)
print(result)
```

Output

```
[100, 255, 255, 0]
bytearray(b'd\xff\xff\x00')
[100, 255, 255, 0]
```

Convert string. A bytearray can be created from a string. The encoding (like "ascii") is specified as the second argument in the bytearray constructor.

Decode:
To convert from a bytearray back into a string, the decode method is needed.

Python program that converts string, bytearray

```
# Create a bytearray from a string with ASCII encoding.
arr = bytearray("abc", "ascii")
print(arr)

# Convert bytearray back into a string.
result = arr.decode("ascii")
print(result)
```

Output

```
bytearray(b'abc')
abc
```

Append, del, insert. A bytearray supports many of the same operations as a list. We can append values. We can delete a value or a range of values with del. And we can insert a value.

Python program that uses append, del, insert

```
# Create bytearray and append integers as bytes.
values = bytearray()
values.append(0)
values.append(1)
values.append(2)
print(values)

# Delete the first element.
del values[0:1]
print(values)

# Insert at index 1 the value 3.
values.insert(1, 3)
print(values)
```

Output

```
bytearray(b'\x00\x01\x02')
bytearray(b'\x01\x02')
bytearray(b'\x01\x03\x02')
```

ValueError. Numbers inserted into a bytearray or bytes object must be between 0 and 255 inclusive. If we try to insert an out-of-range number, we will receive a ValueError.

Python program that causes ValueError

```
# This does not work.
values = bytes([3000, 4000, 5000])
print("Not reached")
```

Output

```
Traceback (most recent call last):
  File "/Users/sam/Documents/test.py", line 4, in <module>
    values = bytes([3000, 4000, 5000])
ValueError: byte must be in range(0, 256)
```

Replace. The buffer protocol supports string-like methods. We can use replace() as on a string. The arguments must be bytes objects—here we use "b" literals.

Python program that uses replace on bytes

```
value = b"aaabbb"

# Use bytes replace method.
result = value.replace(b"bbb", b"ccc")
print(result)
```

Output

```
b'aaaccc'
```

Compare. A "b" literal is a bytes object. We can compare a bytearray or a bytes object with this kind of constant. To compare bytes objects, we use two equals signs.

Note:

Two equals signs compares the individual byte contents, not the identity of the objects.

Python program that compares bytes

```
# Create a bytes object with no "bytes" keyword.
value1 = b"desktop"
print(value1)

# Use bytes keyword.
value2 = bytes(b"desktop")
print(value2)

# Compare two bytes objects.
if value1 == value2:
    print(True)
```

Output

```
b'desktop'
b'desktop'
True
```

Start, end. We can handle bytes objects much like strings. Common methods like startswith and endswith are included. These check the beginning and end parts.

Argument:

The argument to startswith and endswith must be a bytes object. We can use the handy "b" prefix.

Python program that uses startswith, endswith

```
value = b"users"
```

```
# Compare bytes with startswith and endswith.
if value.startswith(b"use"):
    print(True)

if value.endswith(b"s"):
    print(True)
```

Output

```
True
True
```

Split, join. The split and join methods are implemented on bytes objects. Here we handle a simple CSV string in bytes. We separate values based on a comma char.

Python program that uses split, join

```
# A bytes object with comma-separate values.
data = b"cat,dog,fish,bird,true"

# Split on comma-byte.
elements = data.split(b",")

# Print length and list contents.
print(len(elements))
print(elements)

# Combine bytes objects into a single bytes object.
result = b",".join(elements)
print(result)
```

Output

```
5
[b'cat', b'dog', b'fish', b'bird', b'true']
b'cat,dog,fish,bird,true'
```

Memoryview. This is an abstraction that provides buffer interface methods. We can create a memoryview from a bytes object, a bytearray or another type like an array.

Array

Tip:

With memoryview we can separate our code that uses the buffer interface from the actual data type. It is an abstraction.

Python program that uses memoryview

```
view = memoryview(b"abc")

# Print the first element.
print(view[0])

# Print the element count.
print(len(view))

# Convert to a list.
print(view.tolist())
```

Output

```
b'a'
3
[97, 98, 99]
```

Performance. Suppose we want to append 256 values to a list. Bytearray here is faster. So we both improve memory size and reduce time required with bytearray over list.

So:

Bytearray is more complex to handle. It does not support non-ASCII characters or large numeric values.

But:

In many programs where these are not required, bytearray can be used to improve speed. This benchmark supports this idea.

Python program that times list, bytearray appends

```
import time

print(time.time())

# Version 1: append to list.
for i in range(0, 1000000):
    x = list()
    for v in range(0, 255):
        x.append(v)

print(time.time())

# Version 2: append to bytearray.
for i in range(0, 1000000):
    x = bytearray()
    for v in range(0, 255):
        x.append(v)

print(time.time())
```

Results

```
1411859925.29213
1411859927.673053    list append:      2.38 s
1411859929.463818    bytearray append: 1.79 s  [faster]
```

Read bytes from file. A file can be read into a bytes object. We must specify the "b" mode—to read a file as bytes, we use the argument "rb."

File: Read Binary File

Bytes and bytearrays are an efficient, byte-based form of strings. They have many of the same methods as strings, but can also be used as lists.

In Python, lists can become inefficient quickly. And strings, immutable, lead to excessive copying. Where we represent data in bytes, numbers from 0 to 255, these buffer types are ideal.

Dot Net Perls

Written by Sam Allen,
info@dotnetperls.com (Dot Net Perls).