Binary data in Python3

Bytes

The *bytes* type in python3 is an **immutable** sequence of **8-bit integers**.

Creating empty bytes

Python

```python
>>> null_bytes = bytes(5)
>>> null_bytes
b'\x00\x00\x00\x00\x00'
>>> type(null_bytes)
<class 'bytes'>
>>> null_bytes[0]
0
>>> type(null_bytes[0])
<class 'int'>
>>> len(null_bytes)
5
```

Bytearray

The *bytearray* type is the **mutable** version of *bytes*

```
>>> null_bytes = bytearray(5)
>>> null_bytes
bytearray(b'\x00\x00\x00\x00\x00')
>>> type(null_bytes)
<class 'bytearray'>
>>> null_bytes[0]
0
>>> type(null_bytes[0])
<class 'int'>
>>> null_bytes[1] = 65
>>> null_bytes
bytearray(b'\x00A\x00\x00\x00')
>>> null_bytes.append(66)
>>> null_bytes
bytearray(b'\x00A\x00\x00\x00B')
>>> len(null_bytes)
6
```

```
>>> foo = bytearray(b'Python3')
>>> foo[-1] = 36
>>> foo[0] = 0x24
>>> foo[3] = b'W'[0]
>>> foo
bytearray(b'$ytWon$')
```

Bytes Literals

One way of creating a *bytes* object is **bytes literals**.

- They are always prefixed with `b` or `B`
- They must only contain **ascii** characters
- bytes with a value of 128 or greater must be escaped with `\`
- All bytes including **non-pritable** values can be expressed with `\x` scape sequence, which means the next two characters are interpreted as hex digits for the character code

```
>>> foo = b'\x54\x45\x53\x54'
>>> foo
b'TEST'
>>> type(foo)
<class 'bytes'>
```

Creating biggers integer from bytes

from_bytes method of *int* type returns the integer represented by the given array of bytes.

Python
```
>>> foo = b'\x01\x80'
>>> foo
b'\x01\x80'
>>> foo[0]
1
>>> foo[1]
128
>>> n = int.from_bytes(foo, byteorder='big')
>>> n
384
```

Endiness of integers

The *byteorder* argument determines the byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array.

To see the native byte order of the host os, use `sys.byteorder` as the byte order value.

Python
```
>>> int.from_bytes(b'\x01\x80', byteorder='big')
384
>>> int.from_bytes(b'\x01\x80', byteorder='little')
32769
```

Creating bytes object

Bytes from a single of integer

Python
```
>>> n.to_bytes(1, byteorder='big')
b'\x82'
>>> n.to_bytes(10, byteorder='big')
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x82'
```

Bytes from a list of integers

```python
>>> bar = bytes([65,66,67,68,69])
>>> bar
b'ABCDE'
```

List from a bytes object

```python
>>> foo = b'abc'
>>> foo
b'abc'
>>> list(foo)
[97, 98, 99]
```

Bytes from Hex

```python
>>> bytes.fromhex('506573')
b'Pes'
```

Hex from Bytes

```python
>>> foo = b'hello'
>>> foo.hex()
'68656c6c6f'
```

Bytes from String

```python
>>> text = 'TheFox'
>>> text.encode('utf-8')
b'TheFox'
```

Base 2

```python
>>> bar = int('10101010', 2)
>>> bar
170
>>> bin(170)
'0b10101010'
```

Text encoding

Python
```
>>> foo = b'Hello'
>>> bar = foo.decode('utf-8')
>>> foo
b'Hello'
>>> bar
'Hello'
```

Formatting Strings

Python
```
>>> b = b'\xFF'
>>> '{0:b}'.format(b[0])
'11111111'
>>> '{0:x}'.format(b[0])
'ff'
>>> '{0:o}'.format(b[0])
'377'
>>> '{0:d}'.format(b[0])
'255'
```

Bitwise Operators

- `x & y` AND
- `x | y` OR
- `x ^ y` XOR
- `x >> y` Shift right
- `x << y` Shift left

AND/OR

```python
>>> x = int('11110000', 2)
>>> y = int('00001111', 2)
>>> x & y
0
>>> bin(x & y)
'0b0'
>>> x | y
255
>>> bin(x | y)
'0b11111111'
```

XOR

```python
>>> x = int('10000001', 2)
>>> y = int('10101010', 2)
>>> x ^ y
43
>>> bin(x ^ y)
'0b101011'
```

Shift

```python
>>> n = int('00000110', 2)
>>> n
6
>>> n << 2
24
>>> bin(n << 2)
'0b11000'
>>> n >> 1
3
>>> bin(n >> 1)
'0b11'
```