CS2911: Code Reading Practice

The purpose of this document is to provide a variety of simple (yet tricky!) programs to give students practice with reading and executing code by hand. Some of the problems also exercise Python encoding of integers in raw binary or ASCII decimal. If you find these problems interesting, I encourage you to bring your answers to my office for us to discuss them.

1. Consider this code

```
m = b''
for i range(0, 2):
   m = next_byte()
return m
```

   a. ***Write*** the ***bytes object literal*** for the return value of this function when it receives the message with contents. ***Write*** bytes within bytes objects as ASCII characters rather than \x-escaped codes where possible.

   0d 0a 61 62 41

2. Consider this code

```
e = b''
j = b''
while e != b'\n':
   e = next_byte()
   j += next_byte()
return (e,j)
```

b. ***Complete*** the ***literal*** for the return value of this function when it receives the message with contents. (A literal tuple is just written as parenthesis, as shown below. Fill in the two bytes objects returned by this method.) ***Write*** bytes within bytes objects as ASCII characters rather than \x-escaped codes where possible.

61 62 41 0d 0a 41 0d 0a 35 0d 0a

(,)

3. Consider this code

```
e = next_byte()
while e != b'\n':
  e = next_byte()
  j += e
return (e,j)
```

c. ***Write*** the ***literal*** for the return value of this function when it receives the message with contents. (A literal tuple is just written as parenthesis, as shown below. Fill in the two bytes objects returned by this method.) ***Write*** bytes within bytes objects as ASCII characters rather than \x-escaped codes where possible.

46 61 64 65 64 20 62 65 61 64 0d 0a

4. Consider this code

```
e = next_byte()
f = int(e)
g = b''
for i in range(0,f):
  g += e
  e = next_byte()
return (f,g,e)
```

   d. ***Write*** the ***literal*** for the return value of this function when it receives the message with contents. (A literal tuple is just written as parenthesis, as shown below. Fill in the two bytes objects returned by this method.) ***Write*** bytes within bytes objects as ASCII characters rather than \x-escaped codes where possible.

37 61 64 65 64 20 62 66 33 64 0d 0a

5. Consider this code

```
e = b''
while next_byte() != b'\n':
  g += e
  e = next_byte()
return (g,e)
```

   e. ***Write*** the ***literal*** for the return value of this function when it receives the message with contents. (A literal tuple is just written as parenthesis, as shown below. Fill in the two bytes objects returned by this method.) ***Write*** bytes within bytes objects as ASCII characters rather than \x-escaped codes where possible.

41 31 0d 0a 37 0d 0a 35 0d 0a

6. Consider this code

```
b = b''
for i in range(0,2):
   b += next_byte()
return b
```

f. ***Write*** the ***literal*** for the return value of this function when it receives the message with contents. ***Write*** bytes within bytes objects as ASCII characters rather than \x-escaped codes where possible.

0d 10

g. ***Write*** the ***literal*** for the return value of this function when it receives the message with contents. ***Write*** bytes within bytes objects as ASCII characters rather than \x-escaped codes where possible.

31 32 33 34

7. Consider this code

```
b = b''
for i in range(0,4):
   b = next_byte()
return b
```

h. ***Write*** the ***literal*** for the return value of this function when it receives the message with contents. ***Write*** bytes within bytes objects as ASCII characters rather than \x-escaped codes where possible.

31 32 0d 0a 66 61 64 65 64 0d 0a

8. Consider this code

```
f = b''
q = next_byte()
while q != b'A'
   f += q
   q = next_byte()
return (f, q)
```

i. ***Write*** the ***literal*** for the return value of this function when it receives the message with contents. ***Write*** bytes within bytes objects as ASCII characters rather than \x-escaped codes where possible.

0D 0A 43 42 41 63 62 61

j. ***Write*** the ***literal*** for the return value of this function when it receives the message with contents. ***Write*** bytes within bytes objects as ASCII characters rather than \x-escaped codes where possible.

46 61 63 20 41 63

9. Consider this code

```
c = b''
q = next_byte()
while q != b'a':
   q = next_byte()
   c += q
return (c, q)
```

k. ***Write*** the ***literal*** for the return value of this function when it receives the message with contents. ***Write*** bytes within bytes objects as ASCII characters rather than \x-escaped codes where possible.

46 65 65 64 20 61 20 64

10. Consider this code

```
g = next_byte()
g += next_byte()
h = int(g)
j = b''
k = b''
for i in range(0,h):
   j += k
   k = next_byte()
l = int(k)
return l
```

a. On the right above***, trace*** the values of the variables as **literals** as the program runs for the message received below. Display ASCII characters instead of codes where possible.

b. ***Write*** the ***literal*** for the return value of this function when it receives the message with contents. ***Write*** bytes within bytes objects as ASCII characters rather than \x-escaped codes where possible.

32 31 33 35 37

11. Consider this code

```
d = b''
e = b''
while e != b'\x00':
   d += e
   e = next_byte()
return int.from_bytes(d, 'big'()
```

c. On the right above***, trace*** the values of the variables as **literals** as the program runs for the message received below. Display ASCII characters instead of codes where possible.

d. ***Write*** the ***literal*** for the return value of this function when it receives the message with contents.

01 00

12. Consider this code

```
d = b''
e = b''
while e != b'\x00':
   e = next_byte()
   d += e
return int.from_bytes(d, 'big'()
```

e. On the right above***, trace*** the values of the variables as **literals** as the program runs for the message received below. Display ASCII characters instead of codes where possible.

f. ***Write*** the ***literal*** for the return value of this function when it receives the message with contents.

01 00

g. What values can this protocol not send?