CPSC 1301, Computer Science I Lab Assignment

Lab 8a

Run the following commands in your Python interpreter. These are from *Learning Python, 5th Edition*, Chapter09. Submit a transcript of your session as your deliverable for this lab. This should be a plain text file named `cpsc1301lab08a_lastname.txt`.

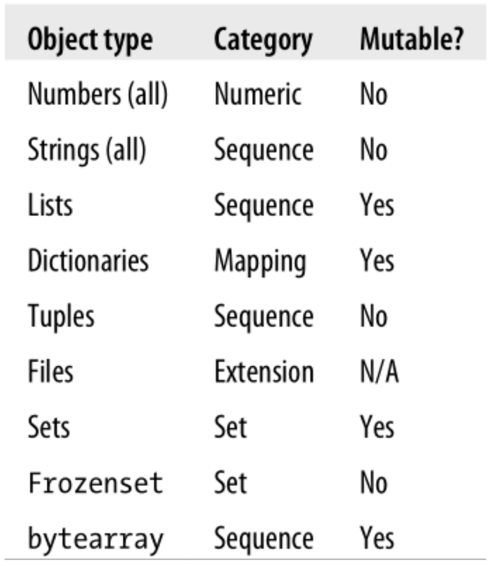
1 Lab

```
%    # sequence operations
>>> (1, 2) + (3, 4) # Concatenation
>>> (1, 2) * 4 # Repetition
>>> T = (1, 2, 3, 4) # Indexing, slicing
>>> T[0], T[1:3]
    # parentheses and commas
>>> x = (40) # An integer!
>>> x
>>> y = (40,) # A tuple
>>> y
    # conversions, methods, immutability
>>> T = ('cc', 'aa', 'dd', 'bb')
>>> tmp = list(T) # Make a list from a tuple's items
>>> tmp.sort() # Sort the list
>>> tmp
>>> T = tuple(tmp) # Make a tuple from the list's items
>>> T
>>> sorted(T) # Or use the sorted built-in, and save two steps
>>> T = (1, 2, 3, 4, 5)
>>> L = [x + 20 for x in T]
>>> L
>>> T = (1, 2, 3, 2, 4, 2) # Tuple methods in 2.6, 3.0, and later
>>> T.index(2) # Offset of first appearance of 2
>>> T.index(2, 2) # Offset of appearance after offset 2
>>> T.count(2) # How many 2s are there?
>>> T = (1, [2, 3], 4)
>>> T[1] = 'spam' # This fails: can't change tuple itself
>>> T[1][0] = 'spam' # This works: can change mutables inside
>>> T
    # records and named tuples
>>> bob = ('Bob', 40.5, ['dev', 'mgr']) # Tuple record
>>> bob
>>> bob[0], bob[2] # Access by position
>>> bob = dict(name='Bob', age=40.5, jobs=['dev', 'mgr']) # Dictionary record
>>> bob
>>> bob['name'], bob['jobs'] # Access by key
>>> list(bob.items()) # Items to tuple list
>>> from collections import namedtuple # Import extension type
>>> Rec = namedtuple('Rec', ['name', 'age', 'jobs']) # Make a generated class
>>> bob = Rec('Bob', age=40.5, jobs=['dev', 'mgr']) # A named-tuple record
>>> bob
>>> bob[0], bob[2] # Access by position
>>> bob.name, bob.jobs # Access by attribute
>>> O = bob._asdict() # Dictionary-like form
>>> O['name'], O['jobs'] # Access by key too
>>> O
>>> name, age, jobs = bob # Tuple assignment (Chapter 11)
>>> name, jobs
>>> for x in bob: print(x) # Iteration context (Chapters 14, 20)
>>>    prints Bob, 40.5, ['dev', 'mgr']...
>>> bob = {'name': 'Bob', 'age': 40.5, 'jobs': ['dev', 'mgr']}
>>> job, name, age = bob.values()
>>> name, job # Dict equivalent (but order may vary)
>>> for x in bob: print(bob[x]) # Step though keys, index values
>>>    prints values...
>>> for x in bob.values(): print(x) # Step through values view
>>>    prints values...
    # files
>>> myfile = open('myfile.txt', 'w') # Open for text output: create/empty
>>> myfile.write('hello text file\n') # Write a line of text: string
>>> myfile.write('goodbye text file\n')
>>> myfile.close() # Flush output buffers to disk
>>> myfile = open('myfile.txt') # Open for text input: 'r' is default
>>> myfile.readline() # Read the lines back
>>> myfile.readline()
>>> myfile.readline() # Empty string: end-of-file
>>> open('myfile.txt').read() # Read all at once into string
>>> print(open('myfile.txt').read()) # User-friendly display
>>> for line in open('myfile.txt'): # Use file iterators, not reads
>>>     print(line, end='')
>>>

>>> f = open("presidents.csv")
>>> for line in f:
>>>     print(line, end = '')
>>>
>>> with open("presidents.csv") as f:
>>>     for line in f:
>>>             print(line, end = '')
>>>
>>> X, Y, Z = 43, 44, 45 # Native Python objects
>>> S = 'Spam' # Must be strings to store in file
>>> D = {'a': 1, 'b': 2}
>>> L = [1, 2, 3]
>>>
>>> F = open('datafile.txt', 'w') # Create output text file
>>> F.write(S + '\n') # Terminate lines with \n
>>> F.write('%s,%s,%s\n' % (X, Y, Z)) # Convert numbers to strings
>>> F.write(str(L) + '$' + str(D) + '\n') # Convert and separate with $
>>> F.close()
>>> chars = open('datafile.txt').read() # Raw string display
>>> chars
>>> print(chars) # User-friendly display
>>> F = open('datafile.txt') # Open again
>>> line = F.readline() # Read one line
>>> line
>>> line.rstrip() # Remove end-of-line
>>> line = F.readline() # Next line from file
>>> line # It's a string here
>>> parts = line.split(',') # Split (parse) on commas
>>> parts
>>> int(parts[1]) # Convert from string to int
>>> numbers = [int(P) for P in parts] # Convert all in list at once
>>> numbers
>>> line
>>> parts = line.split('$') # Split (parse) on $
>>> parts
>>> eval(parts[0]) # Convert to any object type
>>> objects = [eval(P) for P in parts] # Do same for all in list
>>> objects
```

2 Review

Please review and make sure you understand everything in figure 1 except for `bytearray` and `Frozenset`.

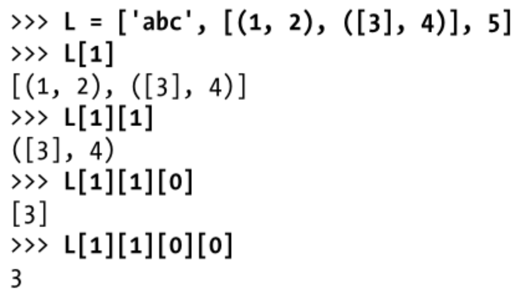


Object type	Category	Mutable?
Numbers (all)	Numeric	No
Strings (all)	Sequence	No
Lists	Sequence	Yes
Dictionaries	Mapping	Yes
Tuples	Sequence	No
Files	Extension	N/A
Sets	Set	Yes
`Frozenset`	Set	No
`bytearray`	Sequence	Yes

Figure 1: Python object classifications

Please review and make sure you understand how everything in figure 2 fits together. It is absolutely critical that you understand how do to this!

Figure 3 contains a pictorial representation of the data structure depicted in figure 2. Trace through this fugure and make sure you understand how all the pieces fit together.



```
>>> L = ['abc', [(1, 2), ([3], 4)], 5]
>>> L[1]
[(1, 2), ([3], 4)]
>>> L[1][1]
([3], 4)
>>> L[1][1][0]
[3]
>>> L[1][1][0][0]
3
```

Figure 2: Nested objects

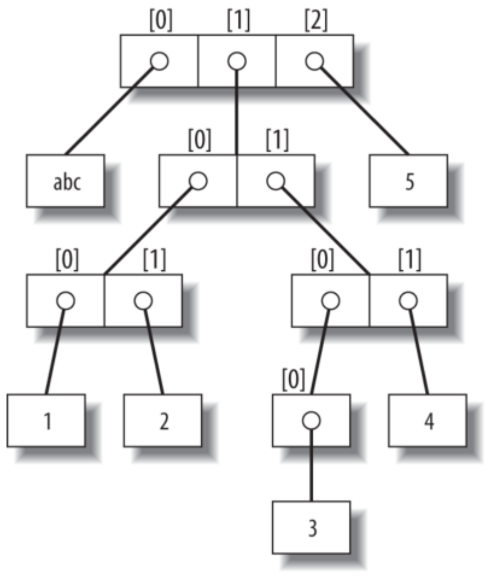


Figure 3: Nested objects picture