# CPSC 1301, Computer Science I Lab Assignment

Lab 07a

Run the following commands in your Python interpreter. These are from *Learning Python, 5th Edition*, Chapter08. Submit a transcript of your session as your deliverable for this lab. This should be a plain text file named `cpsc1301lab07a_lastname.txt`.

> **IMPORTANT:** This chapter is a "must-know" chapter. Reread it carefully. Of all the chapters in the book, this may be the most important, and it certainly is in the top five of the most important chapters.

```
    # Lists
>>> len([1, 2, 3]) # Length
>>> [1, 2, 3] + [4, 5, 6] # Concatenation
>>> ['Ni!'] * 4 # Repetition
>>> str([1, 2]) + "34" # Same as "[1, 2]" + "34"
>>> [1, 2] + list("34") # Same as [1, 2] + ["3", "4"]
    # List operations
>>> 3 in [1, 2, 3] # Membership
>>> for x in [1, 2, 3]:
>>>     print(x, end=' ') # Iteration (2.X uses: print x,)
>>>
>>> res = [c * 4 for c in 'SPAM'] # List comprehensions
>>> res
>>> res = []
>>> for c in 'SPAM': # List comprehension equivalent
>>>     res.append(c * 4)
>>>
>>> res
    # List slicing
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[2] # Offsets start at zero
>>> L[-2] # Negative: count from the right
>>> L[1:] # Slicing fetches sections
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> matrix[1]
>>> matrix[1][1]
>>> matrix[2][0]
>>> matrix = [[1, 2, 3],
>>>     [4, 5, 6],
>>>     [7, 8, 9]]
>>> matrix[1][1]
    # List assignments (changing in place)
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[1] = 'eggs' # Index assignment
>>> L
>>> L[0:2] = ['eat', 'more'] # Slice assignment: delete+insert
```

```
>>> L # Replaces items 0,1
>>> L[1:2] = [4, 5] # Replacement/insertion
>>> L
>>> L[1:1] = [6, 7] # Insertion (replace nothing)
>>> L
>>> L[1:2] = [] # Deletion (insert nothing)
>>> L
>>> L = [1]
>>> L[:0] = [2, 3, 4] # Insert all at :0, an empty slice at front
>>> L
>>> L[len(L):] = [5, 6, 7] # Insert all at len(L):, an empty slice at end
>>> L
>>> L.extend([8, 9, 10]) # Insert all at end, named method
>>> L
    # List methods
>>> L = ['eat', 'more', 'SPAM!']
>>> L.append('please') # Append method call: add item at end
>>> L
>>> L.sort() # Sort list items ('S' < 'e')
>>> L
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort() # Sort with mixed case
>>> L
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort(key=str.lower) # Normalize to lowercase
>>> L
>>>
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort(key=str.lower, reverse=True) # Change sort order
>>> L
>>> L = ['abc', 'ABD', 'aBe']
>>> sorted(L, key=str.lower, reverse=True) # Sorting built-in
>>> L = ['abc', 'ABD', 'aBe']
>>> sorted([x.lower() for x in L], reverse=True) # Pretransform items: differs!
>>> L = [1, 2]
>>> L.extend([3, 4, 5]) # Add many items at end (like in-place +)
>>> L
>>> L.pop() # Delete and return last item (by default: -1)
>>> L
>>> L.reverse() # In-place reversal method
>>> L
>>> list(reversed(L)) # Reversal built-in with a result (iterator)
>>> L = []
>>> L.append(1) # Push onto stack
>>> L.append(2)
>>> L
>>> L.pop() # Pop off stack
>>> L
>>> L = ['spam', 'eggs', 'ham']
>>> L.index('eggs') # Index of an object (search/find)
>>> L.insert(1, 'toast') # Insert at position
>>> L
>>> L.remove('eggs') # Delete by value
>>> L
```

```
>>> L.pop(1) # Delete by position
>>> L
>>> L.count('spam') # Number of occurrences
>>> L = ['spam', 'eggs', 'ham', 'toast']
    # Other list operations
>>> del L[0] # Delete one item
>>> L
>>> del L[1:] # Delete an entire section
>>> L # Same as L[1:] = []
>>> L = ['Already', 'got', 'one']
>>> L[1:] = []
>>> L
>>> L[0] = []
>>> L
    # Dictionaries
>>> D = {'spam': 2, 'ham': 1, 'eggs': 3} # Make a dictionary
>>> D['spam'] # Fetch a value by key
>>> D # Order is "scrambled"
>>> len(D) # Number of entries in dictionary
>>> 'ham' in D # Key membership test alternative
>>> list(D.keys()) # Create a new list of D's keys
    # Dictionary modification
>>> D
>>> D['ham'] = ['grill', 'bake', 'fry'] # Change entry (value=list)
>>> D
>>> del D['eggs'] # Delete entry
>>> D
>>> D['brunch'] = 'Bacon' # Add new entry
>>> D
    # Dictionary methods
>>> D = {'spam': 2, 'ham': 1, 'eggs': 3}
>>> list(D.values())
>>> list(D.items())
>>> D.get('spam') # A key that is there
>>> print(D.get('toast')) # A key that is missing
>>> D.get('toast', 88)
>>> D
>>> D2 = {'toast':4, 'muffin':5} # Lots of delicious scrambled order here
>>> D.update(D2)
>>> D
>>> D.pop('muffin')
>>> D.pop('toast') # Delete and return from a key
>>> D
>>> L = ['aa', 'bb', 'cc', 'dd']
>>> L.pop() # Delete and return from the end
>>> L
>>> L.pop(1) # Delete from a specific position
>>> L
    # Dictionary example
>>> table = {'1975': 'Holy Grail', # Key: Value
>>>     '1979': 'Life of Brian',
>>>     '1983': 'The Meaning of Life'}
>>>
>>> year = '1983'
```

```
>>> movie = table[year] # dictionary[Key] => Value
>>> movie
>>> for year in table: # Same as: for year in table.keys()
>>>     print(year + '\t' + table[year])
>>>
>>> table = {'Holy Grail': '1975', # Key=>Value (title=>year)
>>>     'Life of Brian': '1979',
>>>     'The Meaning of Life': '1983'}
>>>
>>> table['Holy Grail']
>>> list(table.items()) # Value=>Key (year=>title)
>>> [title for
>>> K = 'Holy Grail'
>>> table[K] # Key=>Value (normal usage)
>>> V = '1975'
>>> [key for (key, value) in table.items() if value == V] # Value=>Key
>>> [key for key in table.keys() if table[key] == V] # Ditto
    # Dictionaries with integer keys
>>> L = []
>>> L[99] = 'spam'
>>> D = {}
>>> D[99] = 'spam'
>>> D[99]
>>> D
>>> table = {1975: 'Holy Grail',
>>>     1979: 'Life of Brian', # Keys are integers, not strings
>>>     1983: 'The Meaning of Life'}
>>> table[1975]
>>> list(table.items())
    # Tuple keys
>>> Matrix = {}
>>> Matrix[(2, 3, 4)] = 88
>>> Matrix[(7, 8, 9)] = 99
>>>
>>> X = 2; Y = 3; Z = 4 # ; separates statements: see Chapter 10
>>> Matrix[(X, Y, Z)]
>>> Matrix
>>> Matrix[(2,3,6)]
>>> if (2, 3, 6) in Matrix: # Check for key before fetch
>>>     print(Matrix[(2, 3, 6)]) # See Chapters 10 and 12 for if/else
>>>     else:
>>>     print(0)
>>>
>>> try:
>>>     print(Matrix[(2, 3, 6)]) # Try to index
>>>     except KeyError: # Catch and recover
>>>     print(0) # See Chapters 10 and 34 for try/except
>>>
>>> Matrix.get((2, 3, 4), 0) # Exists: fetch and return
>>> Matrix.get((2, 3, 6), 0) # Doesn't exist: use default arg
    # Dictionary nesting
>>> rec = {}
>>> rec['name'] = 'Bob'
>>> rec['age'] = 40.5
```

```
>>> rec['job'] = 'developer/manager'
>>>
>>> print(rec['name'])
>>> rec = {'name': 'Bob',
>>>        'jobs': ['developer',manager'],
>>>        'web': 'www.bobs.org\~Bob',
>>>        'home': {'state': 'Overworked', 'zip': '12345'}}
>>>
>>> rec['name']
>>> rec['jobs']
>>> rec['jobs'][1]
>>> rec['home']['zip']
    # Making databases
>>> db = []
>>> db.append(rec) # A list "database"
>>> db.append(other)
>>> db[0]['jobs']
>>> db = {}
>>> db['bob'] = rec # A dictionary "database"
    # Optional
>>> D = {'a': 1, 'b': 2, 'c': 3}
>>> D
>>> Ks = D.keys() # Sorting a view object doesn't work!
>>> Ks.sort()
>>> Ks = list(Ks) # Force it to be a list and then sort
>>> Ks.sort()
>>> for k in Ks: print(k, D[k]) # 2.X: omit outer parens in prints
>>>
>>> D
>>> Ks = D.keys() # Or you can use sorted() on the keys
>>> for k in sorted(Ks): print(k, D[k]) # sorted() accepts any iterable
>>>     # sorted() returns its result
>>> D
>>> for k in sorted(D): print(k, D[k]) # dict iterators return keys
>>>
```