Three Pillar Global Inc.

Python Learning Program

Phase 1 (beginner)

A. Topics and reading materials

- Zen of Python (http://www.python.org/dev/peps/pep-0020/)
- Interpreted vs. Compiled
- Control flows
- Data structures
- List comprehensions
- Code structures
- *All the above topics can be found in The Python Tutorial* (http://docs.python.org/tutorial/) *chapters 1 to 7*

B. Check point evaluation

P1. Implement a function that will flatten two lists up to a maximum given depth:

```
def flatten(list_a, list_b, max_depth)
```

P2. Merge 2 objects with any depth (including contained dictionaries, lists, sets, strings, integers, floats). Type mismatches should yield a tuple with the two elements. Examples:

```
a = {'x': [1,2,3], 'y': 1, 'z': set([1,2,3]), 'w': 'qweqwe', 't': {'a': [1, 2]}, 'm': [1]}

b = {'x': [4,5,6], 'y': 4, 'z': set([4,2,3]), 'w': 'asdf', 't': {'a': [3, 2]}, 'm': "wer"}
```

Expected result:

```
{'x': [1,2,3,4,5,6], 'y': 5, 'z': set([1,2,3,4]), 'w': 'qweqweasdf', 't': {'a': [1, 2, 3,
2]}, 'm': ([1], "wer")}
```

P3. Given a file containing a list of dictionaries implement a sorting algorithm (of your choosing) that will sort the list based on the dictionary keys. The dictionary keys will contain alphabetic characters while the values will be integers. The rule for comparing dictionaries between them is:

- if the value of the dictionary with the lowest alphabetic key is lower than the value of the other dictionary with the lowest alphabetic key, then the first dictionary is smaller than the second.
- if the two values specified in the previous rule are equal reapply the algorithm ignoring the current key.

The input is a file containing the list of dictionaries. Each dictionary key value is specified on the same line in the form <key> <whitespace> <value>. Each list item is split by an empty row. The output is a file containing a list of integers specifying the dictionary list in sorted order. Each integer identifies a dictionary in the order they were received in the input file.

Three Pillar Global Inc.

Phase 2 (intermediate)

A. Topics and reading materials

- Mutable vs. Immutable (http://docs.python.org/tutorial/datastructures.html)
- Package vs. Module (http://docs.python.org/tutorial/modules.html)
- Classes (http://docs.python.org/tutorial/classes.html)
- Exceptions (http://docs.python.org/tutorial/errors.html)
- Function decorators (http://docs.python.org/library/functions.html)
- General reading:
    - http://code.google.com/edu/languages/google-python-class/index.html

B. Check point evaluation

P1. Given a dictionary, swap keys <-> values. The program should use some custom logic to check if the swap is possible, but without using try..except constructs or the __hash__() function. Example:

```
Input: {'a': 123, 'b': 456}

Output: {123: 'a', 456: 'b'}

-------------------------------------------------

Input: {'a': (1, 2, [3])}

Output: Swap is not possible
```

P2. Create an automated card dealer for a Texas Hold'em application. It should be able to handle Deck objects, consisting of Cards. Cards can be added or removed from Decks, and Decks can be shuffled and sorted. When dealing cards, each Player receives a Hand consisting of 2 Cards. After all cards are dealt, the Dealer should draw the table Hand of 5 Cards"

P3. Write a function decorator that tracks how long the execution of the wrapped function took. The decorator will log slow function calls including details about the execution time and function name. The decorator should take an optional threshold argument.

Examples:

```
@time_slow
def myfast():
  pass

@time_slow(threshold=0.05)
def myfast():
  pass
```

Three Pillar Global Inc.

Phase 3 (advanced)

A. Topics and reading materials

- Iterators vs. Generators

  PyIterGen.pdf

- itertools
  - http://docs.python.org/library/itertools.html
- functools
  - http://docs.python.org/library/functools.html
- Metaclasses
  - http://docs.python.org/reference/datamodel.html#object.__new__
  - http://docs.python.org/reference/datamodel.html#customizing-class-creation
  - http://www.cafepy.com/article/python_types_and_objects/ - Newstyle classes, type and metaclass intro
  - http://www.cafepy.com/article/python_attributes_and_methods/ - Attributes, descriptors, resolution order (multi inheritance)
  - http://www.vrplumber.com/programming/metaclasses.pdf
  - http://www.vrplumber.com/programming/mcsamples.tar.gz - Code samples for the presentation above
  - http://cleverdevil.org/computing/78/metaclasses-demystified
- Lambda expressions
  - http://diveintopython.net/power_of_introspection/lambda_functions.html
- Class decorators
  - http://www.python.org/dev/peps/pep-3129/ - PEP describing the class decorators
  - http://wiki.python.org/moin/PythonDecoratorLibrary - a comprehensive list of decorator examples
- Debugging
  - http://www.doughellmann.com/PyMOTW/pdb/
  - http://ipython.org/documentation.html

B. Check point evaluation

P1. Given a binary tree encoded as a tuple (node_label, left_node_tuple, right_node_tuple) write an iterator / generator that returns the nodes of the tree in pre-order.
Example:

```
Input: ('b', ('a', None, None), ('z', ('c', None, None), ('zz', None, None)))

Output: b, a, z, c, zz
```

P2. Write a generator that returns all the subsets of a given set. Example:

```
Input: set([1, 2, 3])

Output: set([1, 2, 3]), set([2, 3]), set([1, 3]), set([3]), set([1,
2]), set([2]), set([1]), set([])
```

P3. Implement singleton pattern using meta classes

P4. Review the Django implementation of Models which uses metaclasses extensively and identify alternative ways to implement the same functionality (or similar) using other techniques and paradigms. If you think the current solution is the best approach, please specify the reasons behind this conclusion.