practice1

September 13, 2022

0.1 Counting Treasure

Get the file treasure.py from Bb. It contains a spec for a simple program, including doctests. Run it using python -m doctest treasure.py. Add code to implement the spec. Do not change the doctests. Put your name(s) and ID(s) as shown at the top. Submit treasure.py. This is worth 5%.

[ ]:
```
# I/we declare that this file represents our own work, and that we
# have not seen any work on this assignment done by others, and that
# we have not shown our work to any others.

# Student name(s): Jiarong Li
# Student ID(s): 20230033

# Do not change the formatting above. For multiple names/IDs, use
# commas to separate.


from collections import Counter

def dict_sort(d):
    result = {}
    for k in sorted(d.keys()):
        # in modern Python, dicts remember the order in which their keys
        # were added, and use that order when being printed
        result[k] = d[k]
    return result

def count_treasure(box):
    '''
    While wandering in the fortress of the goblin king, we've discovered a
    box of treasure!

    Count all the treasure and other items in the box and return the
    result as a `dict`, with keys sorted alphabetically.

    `box` is a `dict` specifying the number of each item, eg:
    >>> count_treasure({'coins': 10, 'diamonds': 10})
    {'coins': 10, 'diamonds': 10}
```

```
    The above is a box containing 10 coins and 10 diamonds, so the output
    is as shown.

    A container (the box itself, or a bag, pouch, etc.) can contain other
    containers. Instead of a number, the containers are specified as a
    list, tuple, or similar. We should include the containers in the count:
    >>> count_treasure({'coins': 10,
    ...                 'bags': [{'coins': 2}, {'coins': 5}]})
    {'bags': 2, 'coins': 17}

    Notice the above is a multi-line doctest, using ...

    Containers can be recursive:
    >>> count_treasure({'bags': [{'bags': [{'coins': 10}]}]})
    {'bags': 2, 'coins': 10}

    Here is a bigger example:
    >>> count_treasure({
    ...   'coins': 10,
    ...   'rubies': 10,
    ...   'enchanted pouches': [{
    ...      'coins': 10,
    ...      'rubies': 10,
    ...      'treasure chests': (
    ...         {'coins': 1000},
    ...         {'coins': 1000},
    ...         {'coins': 1000}
    ...      ) # this was a tuple of 3 treasure chests
    ...    }] # this was a list of 1 enchanted pouches
    ... })
    {'coins': 3020, 'enchanted pouches': 1, 'rubies': 20, 'treasure chests': 3}

    If the input is mis-specified, we expect to see an error:
    >>> count_treasure({'bags': (10, 20, 30)})
    Traceback (most recent call last):
    ...
    TypeError: 'int' object is not iterable
    '''

    # HINT: use a `Counter` to store your results while working
    result = Counter()

    ## YOUR CODE HERE

    # HINT: use `dict_sort(result)` at the end to sort and
    # convert to an ordinary `dict`
```

```
    return dict_sort(result)
```

[ ]:

[ ]: