hw00-python-exercises

August 20, 2024

1 Homework: Python exercises

1.1 Name: *Write your name here*

2 Instructions

In this assignment, your core Python skills will be tested by having you complete or write various functions and classes using concepts such as lists, dictionaries, loops, and recursion. Write code in the sections marked with *# YOUR CODE HERE*.

Do not import any libraries, other than the ones already imported in the Utilities secion below!

For each exercise, you will be given a set of instructions and a test function denoted with the prefix `TEST`. The testing function will contain a `todo_check()` function which will give you a rough estimate of whether your code is functioning as excepted. Feel free to write your own tests to make sure your code works with various different inputs.

2.1 Submission

1. Save the notebook.
2. Enter your name in the appropriate markdown cell provided at the top of the notebook.
3. Select `Kernel` -> `Restart Kernel and Run All Cells`. This will restart the kernel and run all cells. Make sure everything runs without errors and double-check the outputs are as you desire!
4. Submit the `.ipynb` notebook file on Canvas.

2.2 Utilities

The code sections below contain the necessary import statements and the helper `todo_check()` function.

[7]:
```python
import os
import gc
import traceback
import warnings
from pdb import set_trace

# Default seed
seed = 0
```

[8]:

```python
class TodoCheckFailed(Exception):
    pass

def todo_check(asserts, mute=False, **kwargs):
    locals().update(kwargs)
    failed_err = "You passed {}/{} and FAILED the following code checks:\n{}"
    failed = ""
    n_failed = 0
    for check, (condi, err) in enumerate(asserts):
        exc_failed = False
        if isinstance(condi, str):
            try:
                passed = eval(condi)
            except Exception:
                exc_failed = True
                n_failed += 1
                failed += f"\nCheck [{check+1}]: Failed to execute check [{check+1}] due to the following error...\n{traceback.format_exc()}"
        elif isinstance(condi, bool):
            passed = condi
        else:
            raise ValueError("asserts must be a list of strings or bools")

        if not exc_failed and not passed:
            n_failed += 1
            failed += f"\nCheck [{check+1}]: Failed\n\tTip: {err}\n"

    if len(failed) != 0:
        passed = len(asserts) - n_failed
        err = failed_err.format(passed, len(asserts), failed)
        raise TodoCheckFailed(err.format(failed))
    if not mute: print("Your code PASSED all the code checks!")
```

3 Python Exercises

3.1 List Centering (10 points)

Complete the `center_list()` function, which should take in a list `x` as input, and subtract the mean of `x` from every element in `x`. Be sure to store the mean of `x` in the variable `mean` and the centered list in the variable `centered_list`. Both `mean` and `centered_list` should be returned! Run the `TEST_center_list()` function to test your code.

Example

The output for the list `[5, 10, 15, 20, 25]` should be a mean of 15 and the new list `[-10.0, -5.0, 0.0, 5.0, 10.0]`

[4]:
```
def center_list(x):
    mean = None
    centered_list = []
    # YOUR CODE HERE




    return mean, centered_list
```

[4]:
```
def TEST_center_list():
    print("{:=^50}".format('Inputs'))
    x = [5, 10, 15, 20, 25]
    print(f"x: {x}")

    print("{:=^50}".format('Outputs'))
    mean, centered_list = center_list(x)
    print(f"Mean: {mean}")
    print(f"Centered list: {centered_list}")

    todo_check([
        ('mean == 15.0', 'The expected output for `mean` is 15',),
        ("centered_list ==  [-10.0, -5.0, 0.0, 5.0, 10.0]", 'The expected␣
 ↪output for `centered_list` is [-10.0, -5.0, 0.0, 5.0, 10.0]')
    ],
    **locals())

TEST_center_list()
```

```
======================Inputs======================
x: [5, 10, 15, 20, 25]
=====================Outputs======================
Mean: 15.0
Centered list: [-10.0, -5.0, 0.0, 5.0, 10.0]
Your code PASSED all the code checks!
```

3.2 Matrix Replacement (10 points)

Complete the `matrix_replacement()` function, which should take in a list of lists `X` (representing a matrix) as input. The goal of the `matrix_replacement()` function is to replace (i.e., filter) all `None` values with the integer value of 0. If you edited `X` directly or created a new matrix with the replaced values, be sure to return it. Run the `TEST_matrix_replacement()` function to test your code.

Example

The output for the list `[[1, None, 2], [None, 3, 1], [5, 6, None]]` should be `[[1, 0, 2], [0, 3, 1], [5, 6, 0]]`

[5]:

```
def matrix_replacement(X):
    # YOUR CODE HERE






    return X
```

[6]:

```
def TEST_matrix_replacement():
    print("{:=^50}".format('Inputs'))
    X = [[1, None, 2],[None, 3, 1],[5, 6, None]]
    print(f"x: {X}")

    print("{:=^50}".format('Outputs'))
    results = matrix_replacement(X)
    print(f"Results: {results}")

    todo_check([
        ('results == [[1, 0, 2], [0, 3, 1], [5, 6, 0]]', 'The expected output␣
 ↪for `results` is [[1, 0, 2], [0, 3, 1], [5, 6, 0]]',),
    ],
    **locals())
TEST_matrix_replacement()
```

```
======================Inputs======================
x: [[1, None, 2], [None, 3, 1], [5, 6, None]]
=====================Outputs======================
Results: [[1, 0, 2], [0, 3, 1], [5, 6, 0]]
Your code PASSED all the code checks!
```

3.3 Histogram (10 points)

Complete the `matrix_histogram(x)` function, which should take in a list `x` as input. The goal of the `make_histogram(x)` function is to output a Python dictionary mapping each unique integer in `x` to the number of times it appears in `x`. Be sure to store the resulting histogram into the `result` variable and to return it. Run the `TEST_make_histogram()` function to test your code.

Example

The output for the list `[1, 2, 1, 3, 2, 1, 6, 2, 6, 1]` should be `{1: 4, 2: 3, 3: 1, 6: 2}`

[6]:

```
def make_histogram(x):
    result = {}
    # YOUR CODE HERE
    return result
```

[8]:

```
def TEST_make_histogram():
    print("{:=^50}".format('Inputs'))
    x = [1, 2, 1, 3, 2, 1, 6, 2, 6, 1]
    print(f"x: {x}")

    print("{:=^50}".format('Outputs'))
    results = make_histogram(x)
    print(f"Result: {results}")
    todo_check([
        ('isinstance(results, dict)', f'The result of `make_histrogram` must be␣
 ↪of type dict, instead got `{type(results)}`'),
        ('results == {1:4, 2:3, 3:1, 6:2}', 'The expected output for `results`␣
 ↪is {1:4, 2:3, 3:1, 6:2}',),
    ],
    **locals())
TEST_make_histogram()
```

```
======================Inputs======================
x: [1, 2, 1, 3, 2, 1, 6, 2, 6, 1]
=====================Outputs======================
Result: {1: 4, 2: 3, 3: 1, 6: 2}
Your code PASSED all the code checks!
```

3.4 Tree traversal (10 points)

Write a Python function `treefun(t)` that takes as input a rooted tree represented as $[root, subtree_1, subtree_2, ..., subtree_n]$ and returns a tuple `(cnodes, leaves)` where `cnodes` is the number of nodes in the tree and `leaves` is a list of all the leaves of the tree, in left to right order.

Examples - Below is an expanded version of the list given as an example to better visualize a tre ecan be represented sequentially in a list (pre-order).

```
[1,
    [2
        [3],
        [4],
    ],
    [5],
    [6,
         [7,
              [8]
         ]
         [9]
    ]
]
```

- The output for `treefun([1, [2, [3], [4]], [5], [6, [7, [8]], [9]]])` should be the tuple `(9, [3, 4, 5, 8, 9])`

[10]:
```
def treefun(t):
    result = (0, [])

    # YOUR CODE HERE



    return result


# This call should return the tuple (9, [3, 4, 5, 8, 9])
treefun([1, [2, [3], [4]], [5], [6, [7, [8]], [9]]])
```

[10]:
```
(0, [])
```

[12]:
```
def TEST_iterate_over_tree():
    print("{:=^50}".format('Inputs'))
    X = [1, [2, [3], [4]], [5], [6, [7, [8]], [9]]]
    print(f"X: {X}")

    print("{:=^50}".format('Outputs'))
    count, leafs = iterate_over_tree(X)
    print(f"Count: {count}")
    print(f"leafs: {leafs}")

    todo_check([
        ('count == 9', 'The expected output for `count` is 9',),
        ('leafs == [3, 4, 5, 8, 9]', 'The expected output for `leafs` is [3, 4,␣
 ↪5, 8, 9]',),
    ],
    **locals())
TEST_iterate_over_tree()
```

```
======================Inputs======================
X: [1, [2, [3], [4]], [5], [6, [7, [8]], [9]]]
=====================Outputs======================
Count: 9
leafs: [3, 4, 5, 8, 9]
Your code PASSED all the code checks!
```

3.5 One-hot encodings (10 points)

Define a function `encoding(data)` that takes as input a list of tuples called `data`, where each tuple contains one or more nonnegative integers. The functions should output a list of one-hot encodings of the tuples in data, one encoding per tuple, by proceeding in two steps: 1) calculate the maximum integer across all the tuples in data, call this M. 2) for each tuple create the corresponding encoding as a list of M + 1 numbers that are all zeros with the exception of the positions contained in the tuple, where it should contain ones.

For example, if `data = [(3, 1, 5), (2, 0), (1, 2)]`, then `encoding(data)` should return `[[0, 1, 0, 1, 0, 1], [1, 0, 1, 0, 0, 0], [0, 1, 1, 0, 0, 0]]`.

Then write a function `most_frequent(data)` that returns the integer that appears the most often in the tuples in `data`. If there are two or more integers that appear most often, return the smallest one. For the example above, it should return `1`.

[ ]:

```python
def encoding(data):
    # YOUR CODE HERE









# The call below should return [[0, 1, 0, 1, 0, 1], [1, 0, 1, 0, 0, 0], [0, 1, 1, 0, 0, 0]]
encoding([(3, 1, 5), (2, 0), (1, 2)])
```

[ ]:

```python
def most_frequent(data):
    # YOUR CODE HERE









# The call below should return 1
most_frequent([(3, 1, 5), (2, 0), (1, 2)])
```

3.6 Working with text files (10 points)

Write a function text_stats(fname) that read a text file **line by line** and returns a tuple containing the following elements:

1. The total number of *lines* in the file.

2. The number of *words* in the file. A *word* is defined as a maximally contiguous sequence of one ore more letters. For example, the string ‘The SARS-Covid-19 pandemic started in 2020’ contains the words ‘The’, ‘SARS’, ‘Covid’, ‘pandemic’, ‘started’, and ‘in’, hence 6 words.
3. A histogram of the word lengths in the file, i.e. a dictionary that maps integers n to the number words of length n that are in the file. The keys n are between 1 and the maximum word length in the file.

My code has 13 lines.

[9]:
```python
def text_stats(fname):
    # YOUR CODE HERE





    return 0, 0, {}

# The call below calls should return
#(102,
# 833,
# {4: 230,
#  2: 145,
#  3: 162,
#  5: 91,
#  6: 47,
#  7: 49,
#  1: 44,
#  9: 8,
#  10: 11,
#  8: 33,
#  12: 5,
#  13: 2,
#  11: 6})
text_stats('../data/thinking-meat.txt')
```

[9]:
```
(0, 0, {})
```