assignment_three

March 4, 2023

0.1 Timothy Miller

0.2 GTECH 73100, Dr. Sun

1 Assignment Three

Adjustments to assignment two

1.1 Modifications

I considered several changes to the previous implementation. First, I examined the list and dictionary construction functions to determine whether any of them would be easy to read and more concise as comprehensions. I decided that only `get_county_totals` would be improved with a list comprehension. Next, I considered whether any of the dictionary constructions in task three would benefit from consolidation into a single constructor. For example, the first and third parts both require counting the total counties in each state. However, I decided that keeping them as separate functions made them easier to understand and maintain- this is worth a little bit of duplication. I also considered reusuing mock data between several tests. However, I once again decided that keeping these tests independent of each other was worth a bit of code duplication.

Finally, I reexamined the use of the built-in sort function when sorting the rankings. The rankings list will only ever have one element out of position. I wanted to check whether this knowledge could lead to a more performant custom sorting algorithm that takes advantage of this structure. I created a 'simple bubble' function to sort the rankings. I then compared its performance to the builtin sort. It's worse. But, now I know.

I only included the portions of code with meaningful changes, to help make them easier to read. Ultimately, these changes were limited to a few parts of Task two. Additionally, there are comments noting the modifications.

1.1.1 Import modules

[ ]:
```
import json
import io
import time
```

1.1.2 Read in a data file of all counties in the US.

[ ]:
```
# Data Source https://eric.clst.org/assets/wiki/uploads/Stuff/
 ↪gz_2010_us_050_00_5m.json
# address utf-8 error https://stackoverflow.com/questions/30996289/
 ↪utf8-codec-cant-decode-byte-0xf3
# Use "with ... as ..." to better handle exceptions
with io.open('data/gz_2010_us_050_00_5m.json', encoding='latin-1') as f:
  data = json.load(f)

features = data['features']
```

1.1.3 Task 2

Derive the numbers of counties that use these three names, respectively. For each of them, list their county name and state code.

[ ]:
```
def get_county_states(features):
  """Format the counties to easily list their states

  Arguments:
  features list[dict] -- county data

  Returns:
  dict[str, list[str]] -- County name as the key and the list of state codes as␣
 ↪the value
  """
  county_states = dict()
  for feature in features:
    properties = feature['properties']
    county_name = properties['NAME']
    state_code = properties['STATE']
    county_states.setdefault(county_name, []).append(state_code)

  return county_states


def test_get_county_states():
  mock_features = [{
    'type': 'Feature',
    "properties": {
      'GEO_ID': '0500000US01087',
      'STATE': '01',
      'COUNTY': '087',
      'NAME': 'Macon',
      'LSAD': 'County',
      'CENSUSAREA': 608.885
    },
```

```
    "geometry": None,
  }, {
    'type': 'Feature',
    "properties": {
      'GEO_ID': '0500000US02275',
      'STATE': '02',
      'COUNTY': '275',
      'NAME': 'Wrangell',
      'LSAD': 'Cty&Bor',
      'CENSUSAREA': 2541.483
    },
    "geometry": None,
  }, {
    'type': 'Feature',
      'properties': {
      'GEO_ID': '0500000US02270',
      'STATE': '02',
      'COUNTY': '270',
      'NAME': 'Wade Hampton',
      'LSAD': 'CA',
      'CENSUSAREA': 17081.433
    },
    "geometry": None,
  }, {
    'type': 'Feature',
      'properties': {
      'GEO_ID': '',
      'STATE': '03',
      'COUNTY': '',
      'NAME': 'Wade Hampton',
      'LSAD': '',
      'CENSUSAREA': 0
    },
    "geometry": None,
  }]

  expected = {
    "Macon": ["01"],
    "Wrangell": ["02"],
    "Wade Hampton": ["02", "03"]
  }

  obs = get_county_states(mock_features)
  assert(obs == expected)

test_get_county_states()
```

```
county_states = get_county_states(features)

def get_county_totals(county_states):
  """Reformat the counties and state list objects into a tuple of counties and␣
 ↪their totals

  Arguments:
  county_states dict[str, list[str]] -- County name as the key and the list of␣
 ↪state codes as the value

  Returns:
  tuple[str, int] -- The counties and the total number of states that use them.
  """
  # Modified to use list comprehension
  return [(county, len(states)) for county, states in county_states.items()]


def test_get_county_totals():
  mock_county_states = {
    "A": ['0'],
    "B": ['0', '1'],
    "C": ['0', '1', '2']
  }
  expected_count_totals = [('A', 1), ('B', 2), ('C', 3)]
  assert(get_county_totals(mock_county_states) == expected_count_totals)

test_get_county_totals()

# Modification: create simple bubble sort
def simple_bubble(totals):
  """Sorting based on the assumption that only the first item needs to be␣
 ↪'bubbled up' to its correct space

  Arguments:
  totals list[tuple[str, int]] -- Item Name and total

  Returns:
  None: list is mutated in place
  """

  target_pos = 0
  compare_pos = 1
  totals_count = len(totals)
  while compare_pos < totals_count and totals[target_pos][1] >␣
 ↪totals[compare_pos][1]:
    totals[target_pos], totals[compare_pos] = totals[compare_pos],␣
 ↪totals[target_pos]
```

```
    target_pos += 1
    compare_pos += 1

def test_simple_bubble():
    mock_totals = [("C", 5), ("B", 0), ("A", 5)]
    expected_totals = [("B", 0), ("C", 5), ("A", 5)]

    # Sort in place
    simple_bubble(mock_totals)

    assert(mock_totals == expected_totals)

test_simple_bubble()

# Modification: create function that sorts with a simple bubble sort
def top_k_sort_k_bubble(totals, k):
  """Find the top k values in a tuple of objects and their counts.

  Arguments:
  totals tuple[str, int]-- Item Name and total
  k int -- the number of items to rank

  Returns:
  tuple[str, int] -- Top k items and their counts

  Note:
  This function iterates through the list of items, only sorting the list of␣
 ↪rankings
  """
  top_k = [(None, 0)]*k
  for total in totals:
    total_val = total[1]
    bottom_k_val = top_k[0][1]
    if total_val > bottom_k_val:
      top_k[0] = total
      simple_bubble(top_k)

  return top_k

def test_top_k_sort_k_simple():
  mock_county_totals = [("A", 0), ("B", 1), ("C", 2), ("D", 3), ("E", 4), ("F",␣
 ↪4)]
  expected_top_k = [("D", 3), ("F", 4), ("E", 4)]

  obs = top_k_sort_k_bubble(mock_county_totals, 3)

  assert(obs == expected_top_k)
```

```
test_top_k_sort_k_simple()

def top_k_sort_k(totals, k):
  """Find the top k values in a tuple of objects and their counts.

  Arguments:
  totals tuple[str, int]-- Item Name and total
  k int -- the number of items to rank

  Returns:
  tuple[str, int] -- Top k items and their counts

  Note:
  This function iterates through the list of items, only sorting the list of␣
 ↪rankings
  """
  top_k = [(None, 0)]*k
  for total in totals:
    total_val = total[1]
    bottom_k_val = top_k[0][1]
    if total_val > bottom_k_val:
      top_k[0] = total
      top_k.sort(key=lambda a: a[1])

  return top_k

def test_top_k_sort_k():
  mock_county_totals = [("A", 0), ("B", 1), ("C", 2), ("D", 3), ("E", 4), ("F",␣
 ↪4)]
  expected_top_k = [("D", 3), ("F", 4), ("E", 4)]

  assert(top_k_sort_k(mock_county_totals, 3) == expected_top_k)

test_top_k_sort_k()

county_totals = get_county_totals(county_states)
# Modification: Compare the previous sort of the rankings with the
# new method that uses a simple bubble
"""
Compare the built-in and simple bumble sort algorithm approaches

Findings: Built in sort is almost always faster
"""
k = 200
start_time = time.perf_counter()
top_counties = top_k_sort_k(county_totals, k)
```

```
stop_time = time.perf_counter()
print(f"top counties built in: {top_counties}")
print(f"top counter built in time: {stop_time - start_time}")

start_time = time.perf_counter()
top_counties_bubble = top_k_sort_k_bubble(county_totals, k)
stop_time = time.perf_counter()
print(f"top counties bubble: {top_counties_bubble}")
print(f"top counter bubble time: {stop_time - start_time}")
```

```
top counties built in: [('Vernon', 3), ('Choctaw', 3), ('Ohio', 3), ('Lowndes',
3), ('Murray', 3), ('Cheyenne', 3), ('Hall', 3), ('Buchanan', 3), ('Grayson',
3), ('Gallatin', 3), ('Delta', 3), ('Pawnee', 3), ('Morris', 3), ('Meade', 3),
('Daviess', 3), ('Butte', 3), ('Holmes', 3), ('Seminole', 3), ('Summit', 3),
('Park', 3), ('Baker', 3), ('Sussex', 3), ('Miller', 3), ('Stone', 3),
('Sevier', 3), ('Platte', 3), ('Chippewa', 3), ('Williamson', 3), ('Moore', 3),
('Haskell', 3), ('Fairfield', 3), ('Chester', 3), ('Wyoming', 3), ('Cameron',
3), ('Beaver', 3), ('Stephens', 3), ('McIntosh', 3), ('Potter', 3), ('Buffalo',
3), ('Orleans', 3), ('Oneida', 3), ('Miami', 3), ('Burke', 3), ('Schuyler', 3),
('Christian', 3), ('Lauderdale', 3), ('Rock', 3), ('McPherson', 3),
('Covington', 3), ('Todd', 3), ('St. Louis', 3), ('Pope', 3), ('Osceola', 3),
('Thomas', 3), ('Stevens', 3), ('Osage', 3), ('Edwards', 3), ('Dickinson', 3),
('Comanche', 3), ('Wright', 3), ('Worth', 3), ('Humboldt', 3), ('Claiborne', 3),
('Robertson', 3), ('Nelson', 3), ('McLean', 3), ('Teton', 3), ('Walker', 3),
('Pickens', 3), ('Cedar', 3), ('Noble', 3), ('Stark', 3), ('Coffee', 3),
('Cook', 3), ('Kiowa', 3), ('Cleveland', 3), ('Graham', 3), ('Wood', 4),
('Carbon', 4), ('Fremont', 4), ('Ottawa', 4), ('Russell', 4), ('Camden', 4),
('Middlesex', 4), ('Bedford', 4), ('St. Clair', 4), ('Iron', 4), ('Smith', 4),
('Somerset', 4), ('Wheeler', 4), ('Sherman', 4), ('Lancaster', 4), ('Dawson',
4), ('Blaine', 4), ('Dodge', 4), ('Wilson', 4), ('Linn', 4), ('Grundy', 4),
('Adair', 4), ('Sumter', 4), ('Lamar', 4), ('Van Buren', 4), ('Liberty', 4),
('Phillips', 4), ('Mineral', 4), ('San Juan', 5), ('Dallas', 5), ('Richmond',
5), ('Essex', 5), ('Campbell', 5), ('Houston', 5), ('Henderson', 5), ('Carter',
5), ('Caldwell', 5), ('York', 5), ('Pierce', 5), ('Sheridan', 5), ('Kent', 5),
('Lyon', 5), ('Mitchell', 5), ('Allen', 5), ('Anderson', 5), ('White', 5),
('Clarke', 5), ('Decatur', 5), ('Saline', 5), ('Garfield', 6), ('Floyd', 6),
('Mason', 6), ('Custer', 6), ('Sullivan', 6), ('Livingston', 6), ('Jones', 6),
('DeKalb', 6), ('Newton', 6), ('Martin', 6), ('Delaware', 6), ('Hardin', 6),
('Macon', 6), ('Lafayette', 6), ('Lewis', 7), ('Howard', 7), ('Pulaski', 7),
('Richland', 7), ('Taylor', 7), ('Butler', 8), ('Mercer', 8), ('Harrison', 8),
('Cumberland', 8), ('Webster', 8), ('Cherokee', 8), ('Orange', 8), ('Fulton',
8), ('Boone', 8), ('Jasper', 8), ('Randolph', 8), ('Columbia', 8), ('Knox', 9),
('Brown', 9), ('Shelby', 9), ('Clinton', 9), ('Cass', 9), ('Benton', 9),
('Putnam', 9), ('Perry', 10), ('Hancock', 10), ('Logan', 10), ('Hamilton', 10),
('Henry', 10), ('Pike', 10), ('Crawford', 11), ('Lawrence', 11), ('Fayette',
11), ('Scott', 11), ('Morgan', 11), ('Calhoun', 11), ('Lake', 12), ('Clark',
12), ('Marshall', 12), ('Douglas', 12), ('Polk', 12), ('Lee', 12), ('Johnson',
12), ('Adams', 12), ('Carroll', 13), ('Greene', 14), ('Warren', 14), ('Grant',
```

```
15), ('Wayne', 16), ('Monroe', 17), ('Marion', 17), ('Clay', 18), ('Union', 18),
('Montgomery', 18), ('Madison', 20), ('Lincoln', 24), ('Jackson', 24),
('Jefferson', 26), ('Franklin', 26), ('Washington', 31)]
top counter built in time: 0.002795975000026374
top counties bubble: [('Vernon', 3), ('Choctaw', 3), ('Ohio', 3), ('Lowndes',
3), ('Murray', 3), ('Cheyenne', 3), ('Hall', 3), ('Buchanan', 3), ('Grayson',
3), ('Gallatin', 3), ('Delta', 3), ('Pawnee', 3), ('Morris', 3), ('Meade', 3),
('Daviess', 3), ('Butte', 3), ('Holmes', 3), ('Seminole', 3), ('Summit', 3),
('Park', 3), ('Baker', 3), ('Sussex', 3), ('Miller', 3), ('Stone', 3),
('Sevier', 3), ('Platte', 3), ('Chippewa', 3), ('Williamson', 3), ('Moore', 3),
('Haskell', 3), ('Fairfield', 3), ('Chester', 3), ('Wyoming', 3), ('Cameron',
3), ('Beaver', 3), ('Stephens', 3), ('McIntosh', 3), ('Potter', 3), ('Buffalo',
3), ('Orleans', 3), ('Oneida', 3), ('Miami', 3), ('Burke', 3), ('Schuyler', 3),
('Christian', 3), ('Lauderdale', 3), ('Rock', 3), ('McPherson', 3),
('Covington', 3), ('Todd', 3), ('St. Louis', 3), ('Pope', 3), ('Osceola', 3),
('Thomas', 3), ('Stevens', 3), ('Osage', 3), ('Edwards', 3), ('Dickinson', 3),
('Comanche', 3), ('Wright', 3), ('Worth', 3), ('Humboldt', 3), ('Claiborne', 3),
('Robertson', 3), ('Nelson', 3), ('McLean', 3), ('Teton', 3), ('Walker', 3),
('Pickens', 3), ('Cedar', 3), ('Noble', 3), ('Stark', 3), ('Coffee', 3),
('Cook', 3), ('Kiowa', 3), ('Cleveland', 3), ('Graham', 3), ('Wood', 4),
('Carbon', 4), ('Fremont', 4), ('Ottawa', 4), ('Russell', 4), ('Camden', 4),
('Middlesex', 4), ('Bedford', 4), ('St. Clair', 4), ('Iron', 4), ('Smith', 4),
('Somerset', 4), ('Wheeler', 4), ('Sherman', 4), ('Lancaster', 4), ('Dawson',
4), ('Blaine', 4), ('Dodge', 4), ('Wilson', 4), ('Linn', 4), ('Grundy', 4),
('Adair', 4), ('Sumter', 4), ('Lamar', 4), ('Van Buren', 4), ('Liberty', 4),
('Phillips', 4), ('Mineral', 4), ('San Juan', 5), ('Dallas', 5), ('Richmond',
5), ('Essex', 5), ('Campbell', 5), ('Houston', 5), ('Henderson', 5), ('Carter',
5), ('Caldwell', 5), ('York', 5), ('Pierce', 5), ('Sheridan', 5), ('Kent', 5),
('Lyon', 5), ('Mitchell', 5), ('Allen', 5), ('Anderson', 5), ('White', 5),
('Clarke', 5), ('Decatur', 5), ('Saline', 5), ('Garfield', 6), ('Floyd', 6),
('Mason', 6), ('Custer', 6), ('Sullivan', 6), ('Livingston', 6), ('Jones', 6),
('DeKalb', 6), ('Newton', 6), ('Martin', 6), ('Delaware', 6), ('Hardin', 6),
('Macon', 6), ('Lafayette', 6), ('Lewis', 7), ('Howard', 7), ('Pulaski', 7),
('Richland', 7), ('Taylor', 7), ('Butler', 8), ('Mercer', 8), ('Harrison', 8),
('Cumberland', 8), ('Webster', 8), ('Cherokee', 8), ('Orange', 8), ('Fulton',
8), ('Boone', 8), ('Jasper', 8), ('Randolph', 8), ('Columbia', 8), ('Knox', 9),
('Brown', 9), ('Shelby', 9), ('Clinton', 9), ('Cass', 9), ('Benton', 9),
('Putnam', 9), ('Perry', 10), ('Hancock', 10), ('Logan', 10), ('Hamilton', 10),
('Henry', 10), ('Pike', 10), ('Crawford', 11), ('Lawrence', 11), ('Fayette',
11), ('Scott', 11), ('Morgan', 11), ('Calhoun', 11), ('Lake', 12), ('Clark',
12), ('Marshall', 12), ('Douglas', 12), ('Polk', 12), ('Lee', 12), ('Johnson',
12), ('Adams', 12), ('Carroll', 13), ('Greene', 14), ('Warren', 14), ('Grant',
15), ('Wayne', 16), ('Monroe', 17), ('Marion', 17), ('Clay', 18), ('Union', 18),
('Montgomery', 18), ('Madison', 20), ('Lincoln', 24), ('Jackson', 24),
('Jefferson', 26), ('Franklin', 26), ('Washington', 31)]
top counter bubble time: 0.004872815999988234
```