Exercise Set 3

ex_3_0.py

In this module you will implement a `unittest` test cases for the `in_range()` function from `ex_1_0.py`. Create a Python module named `ex_3_0.py`. In order to test the `in_range` function you will need to:

- import `unittest`
- import the `in_range` function from `ex_1_0.py`
- Create a `TestInRange` class that inherits from (subclasses) `unittest.TestCase`
- Create the necessary instance methods to test the minimum number of cases for `in_range` (there should be three).
- implement an entry point and inside the entry point include the call `unittest.main()` so that you can test your test cases.

Reference the Module 3.0 slides for examples on writing `TestCase` class and note that the `assertEqual` methods will be enough to get your test cases.

Note: You will not need to submit ex_1_0.py to CodePost. Your submitted code will have access to another copy for tests

ex_3_1.py

In this exercise you will create a new module `ex_3_1.py` in which you refactor the `user_list()` function from `ex_2_3.py` so that it stores each record in a **collections.namedtuple (https://docs.python.org/3/library/collections.html#namedtuple-factory-function-for-tuples-with-named-fields)** type.

Your `user_list` function should accept a list of lists in the form

```
accounts = [
    ['Sly', 'Brockbank', 'sbrockbank0@patch.com', '118', '5/21/2021'],
    ['Modesta', 'Petegre', 'mpetegre1@kickstarter.com', '135', '10/12/2020']
]
```

where each record has the following fields.

```
fields = ['first_name', 'last_name', 'email', 'login_count', 'last_login']
```

In your module, define the `namedtuple` type with the fields above and a type name of `User`. NOTE: You do *NOT* have to include the username field in the namedtuple or the output records.

Your function should meet the following requirements:

- accept a list of lists of the form above
- return a list of `namedtuple` types

Hint: Use the unpacking operator `*` to easily create a `User` namedtuple from each record in the list.

ex_3_2.py

Consider the hypothetical (but practical) situation where you are tasked with submitting IP addresses collected from access logs to an IP Geolocation API to determine the location of the origin IP. The service charges per address and the true access logs contain 1000's of addresses–some of which are redundant. In order to minimize cost of the API call, it is best to remove any duplicates from the logs of IP addresses.

You are provided with two small samples of IPv4 addresses taken from server access logs–`add_log_1` and `add_log_2`. These are located in the `ip_log.py` module so that you can easily import them into your module. (**Download ip_log.py here (https://cbu.instructure.com/courses/9073/files/1179736?wrap=1)** **(https://cbu.instructure.com/courses/9073/files/1179736/download?download_frd=1)**)

Here is a sample of `add_log_1`:

```
add_log_1 = ['96.236.60.124',
             '186.186.137.38',
             '7.90.250.146',
             '7.90.250.146',
             '7.90.250.146',
             '96.236.60.124',
             # ...
```

Download the support file `ip_log.py` and save it to your project directory.

In a module named `ex_3_2.py` implement a function `get_addresses(*args)` that accepts a variable number of lists of the form shown above and returns a single list with duplicates removed. Import the `add_log_1` and `add_log_2` variables into your module for testing purposes. The unit tests will pass different lists of the same form to your function.

Hints: - there is a built-in data type that will greatly simplify your implementation - the `list.extend()` method is a nice way to accumulate multiple lists into a single list.

ex_3_3.py

In a module named `ex_3_3.py` implement a function `top_3_ips(*args)` that accepts a variable number of lists as positional arguments. The lists are of the same form as in `ex_3_2`. Your function should return the 3 *most common* IP addresses in all input lists *as a list of strings. Example:*

[`'186.186.137.38', '7.90.250.146', '7.90.250.146',]`

Hints: - the **collections (https://docs.python.org/3/library/collections.html#module-collections)** module from the standard library has a class that will greatly simplify your implementation.

ex_3_4.py

This exercise is an extension of `ex_3_2`. In this exercise, you will write a Python program that opens and reads IP address from a file named `ip_log.txt` which contains one IP address per line.

Download the support file `ip_log.txt` (https://cbu.instructure.com/courses/9073/files/1179737?wrap=1) ↓ **(https://cbu.instructure.com/courses/9073/files/1179737/download?download_frd=1)** and save it to your project directory.

In your module, implement the following tasks:

- Open the file `ip_log.txt` for reading.
- Each line in the file lists an IP address. Read all lines into a data container appropriate for passing to your `get_addresses()` function from `ex_3_2`.
- Print the number of unique IP addresses in the log. (Note: print *only* the number. The unittest will look for a single integer string output)
- The unit test will run your code the same way that IDLE does when you choose Run -> Run Module. There is no need to implement a function for this one.

Note: A file with name `ip_log.txt` will be available to your code in CodePost. There is no need to submit the file along with your code.