Exercise Set 2

Submitting

Submit each of your files to the **Exercise Set 2 assignment in CodePost (https://codepost.io)**. Note that if you are submitting them one file at a time, tests for other files will naturally fail. You will still be able to view the test results for the files that you have submitted.

ex_2_0.py

Create a Python module named `ex_2_0.py`. Within that module, implement a function `filter_positive_even()` that meets the following requirements:

- accepts an input list of integers
- filters the input list to produce a new list of positive and even numbers
- returns the filtered list

Hint: It may be helpful to implement a utility function that returns `True` if a number is both positive and even.

ex_2_1.py

In a Python module named `ex_2_1.py`, implement a function `basename()`. `basename()` should take in one or more filenames (as `str` types) and return the basename of the file. To be clear, the basename of a file is the filename without the extension (Ex: `basename('birthday.jpg')` -> `'birthday'`).

Your implementation of `basename()` should meet the following requirements:

- It should not rely on any imported modules.
- Your function should accept a variable number of positional arguments.
  - If one filename is supplied, the function should simply return the basename as a string
  - If more than one filename is supplied, the function should return a list where the i-th element of the list is the basename for the i-th argument.
- For this exercise, it is safe to assume that the caller of the function *will* pass valid filenames.

Here's are some example runs. Note that `basename()` accepts **filenames** not paths.

```
>>> basename('foo.txt')
'foo'
>>> basename('foo.txt', 'test.jpg')
['foo', 'test']
>>> basename('foo.txt', 'test.jpg', 'ex_2_0.py')
['foo', 'test', 'ex_2_0']
>>> files = ['foo.txt', 'test.jpg', 'ex_2_0.py']
>>> basename(*files)
['foo', 'test', 'ex_2_0']
>>>
```

ex_2_2.py

For module `ex_2_2.py` implement a Python class called `User`. This class will represent a user record similar to those used in Exercise Set 1. The class should contain the following attributes and methods:

Attributes defined in the contstructor:

- `first_name`: The first name of the User instance
- `last_name`: The last name for the User represented by the User instance object.
- `username`: A string containing the User instance username
- `email`: A string containing the User instance object's email address
- `last_login`: A string containing the last login date for an instance of the User class.
- `login_count`: An integer storing the number of logins for an instance of the User class.

Instance Methods:

- `__init__()`: This method should be accept positional arguments for each of the attributes listed above and set the correct values for an instance of the User class.
- `__str__()`: This method should return a string with all properties formatted as a dictionary literal: `{'first_name': 'UserFirst', 'last_name': 'UserLast', 'username': 'user', 'email': 'user@example.com', 'last_login': '10/22/2021', 'login_count': 42}`. Hint: Python built-in **vars()** can be used to return a dictionary of instance attributes for an object. You can all the `str()` function on this dictionary to get a string representation of this dictionary.
- `to_dict()`: This instance method should return a dictionary containing the attributes of the User instance (see hint above).

ex_2_3.py

Create a module `ex_2_3.py` in the same directory as `ex_2_2.py`. In this module create a simple utility function called `user_list(users)` that takes in a list of the following form and returns a list of `User` objects based on the input data.

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

Note that the ordered fields are listed in the following list:

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

To accomplish this, your module will need to:

- import your `User` class from the `ex_2_2` module.
- import your `convert_list()` function from your `hp_1` module
- Call `convert_list()` to convert the input list to a list of dictionaries
- …then create and return a list of `User` objects created from each dictionary in the list created using `convert_list()`. (Hint: this should be straightforward since your `User` constructor can be called with keyword-value arguments. Just remember to use dictionary unpacking.)

- to test your function, you can import the `accounts` variable from `hp_1` and call `user_list()` using the accounts variable.

For the `accounts` variable in `hp_1`, the return of your `user_list()` function should look like this:

```
[<ex_2_2.User object at 0x1096eda30>,
 <ex_2_2.User object at 0x1097d3f70>,
 <ex_2_2.User object at 0x1097a6760>,
 <ex_2_2.User object at 0x109883d00>,
 <ex_2_2.User object at 0x109883be0>,
 <ex_2_2.User object at 0x109883cd0>,
 <ex_2_2.User object at 0x109883dc0>,
 <ex_2_2.User object at 0x109883df0>,
 <ex_2_2.User object at 0x109759940>,
 <ex_2_2.User object at 0x1097593d0>]
```

And if you print each item in the list in a loop, you should see dictionary representations for each record in the list.