IS 271 Assignment 2

Submission

Create a file called is271_wk2_LAST_FIRST.py, replacing LAST and FIRST with your name. You will write all your code for this assignment in this file and submit it.

Tasks

Task 1a:

- Write a function named **`day_name`** that takes an integer parameter **`num`** that represents the day of the week.
- Using if/elif/else blocks, return the name of the day of the week.
- Assume Sunday is the first day of the week and it is represented by the number 1.

Example output:

```
In [2]: day_name(1)
Out[2]: 'Sunday'
```

```
In [3]: day_name(7)
Out[3]: 'Saturday'
```

Task 1b:

- Write a function named **`day_name_2`** that performs the same as day_name, but without using if/elif/else blocks.
- *Hint:* Use a list called **`days`** that stores the names of the week in order. Then use the number passed in to get the day at that position of the list.
  - Don't forget to convert from 1 based number to a zero based number!

Task 2:

- Write a function called **`weekend`** that takes the name of a day and returns **`True`** if is a weekend day (Saturday or Sunday) and **`False`** if not.

Example output:

```
In [5]: weekend('Saturday')
Out[5]: True
```

```
In [6]: weekend('Tuesday')
Out[6]: False
```

Task 3a:

- Write a function **`max_while`** that takes a list of integers and uses a **`while`** loop to find the highest value in the list.

Example output:

```
In [8]: max_while([1,2,3,4,5])
Out[8]: 5
```

```
In [9]: max_while([101, 234, 15, 372, 20, 0])
Out[9]: 372
```

Task 3b:

- Write a function **`max_for`** that performs the same behavior as **`max_while`** but uses a **`for`** loop instead.

```
In [10]: #HIDE
def calculate_fine(days):
    return days * 0.25

def balance_due(loans):
    total = 0.0
    for loan in loans:
        total += calculate_fine(loan['days_overdue'])
    return total
```

Task 4a:

- Write a function **`calculate_fine`** that takes an integer representing the number of days overdue
- Make it return the total due in dollars.

- The charge is 25 cents per day.

Example output:

```
In [11]: calculate_fine(5)
Out[11]: 1.25
```

```
In [12]: calculate_fine(0)
Out[12]: 0.0
```

Task 4b:

- Write a function **`balance_due`** that takes a list of dictionaries representing library loans.
- Return the total of all the fines.
- Assume the list of dictionaries take the following structure:

```
[
  {
    "book_ID": "some number"
    "title": "some title",
    "author": "some person",
    "days overdue": 12
  },
  {
    "book_ID": "some number"
    "title": "some title",
    "author": "some person",
    "days overdue": 3
  },
  {
    "book_ID": "some number"
    "title": "some title",
    "author": "some person",
    "days overdue": 0
  }
]
```

- Use your function **`calculate_fine`** from 4a in the code for **`balance_due`**.

Example output:

```
In [13]: my_loans = [
             {'book_ID': 1234, 'title': 'All About A', 'author': 'Mrs. A', 'days_overdue': 10},
             {'book_ID': 1235, 'title': 'Boundless B', 'author': 'Mrs. B', 'days_overdue': 20},
             {'book_ID': 1236, 'title': 'Calling C', 'author': 'Mrs. C', 'days_overdue': 15},
             {'book_ID': 1237, 'title': 'Dreaming D', 'author': 'Mrs. D', 'days_overdue': 30},
         ]

         balance_due(my_loans)
Out[13]: 18.75
```

Task 4c:

- Write a function **`display_loans`** that takes the list of loans and displays the fine for each, followed by the total.
- Present in this format:

```
TITLE: title
AUTHOR: author
DAYS OVERDUE: days overdue
FINE: fine

TITLE: title
AUTHOR: author
DAYS OVERDUE: days overdue
FINE: fine

TITLE: title
AUTHOR: author
DAYS OVERDUE: days overdue
FINE: fine

TOTAL DUE: balance
```

- Use **`calculate_fine`** to get the fine for each item

- Use **balance_due** to calculate the total

Task 4d: Now we will try out some *list comprehensions*.

- Assume we have several library patrons with a list of loans.
- Each patron is represented by a dictionary with some basic info about them and a list of their loans like above.
- Write a function **heavy_users** that takes a list of the patron dictionaries and uses a list comprehension to return a new list of names of users that have more than 10 books on loan.

In [14]:
```
# I'll do this one for you as an example
def heavy_users(patrons):
    return [p['name'] for p in patrons if len(p['loans']) >= 12]
```

In [15]:
```
'''
Notice that the lists of books on loan contain empty dictionaries.
That's OK, because our list comprehension doesn't care what's in them.
It's only concerned with the count
'''
patrons = [
    {'name': 'person A',
     'loans': [{}, {}, {}, {}, {}]},
    {'name': 'person B',
     'loans': [{}, {}, {}, {}, {}, {}, {}, {}, {}, {}]},
    {'name': 'person C',
     'loans': [{}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}]},
    {'name': 'person D',
     'loans': [{}, {}, {}, {}, {}]},
    {'name': 'person E',
     'loans': [{}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}]},
    {'name': 'person F',
     'loans': [{}, {}, {}, {}, {}]},
]

heavy_users(patrons)
```

Out[15]:
```
['person C', 'person E']
```

Task 4e: OK, now it's your turn.

- Write a function **block_list** that takes the same list of patron data.
- Use a list comprehension to return a list of names of patrons that owe $25 or more.
- Use your **balance_due** function from above inside the list comprehension

Example output:

In [17]:
```
patrons = [
    {'name': 'person A',
     'loans': [{'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 10}]},
    {'name': 'person B',
     'loans': [{'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 100},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0}]},
    {'name': 'person C',
     'loans': [{'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0}]},
    {'name': 'person D',
     'loans': [{'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 10}]},
    {'name': 'person E',
     'loans': [{'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 20},
               {'days_overdue': 20},
               {'days_overdue': 20},
               {'days_overdue': 20}]},
    {'name': 'person F',
     'loans': [{'days_overdue': 10},
               {'days_overdue': 10},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0},
               {'days_overdue': 0}]},
]

block_list(patrons)
```

```
Out[17]: ['person B', 'person E']
```

Task 5a:

- Write a function **`parse_data`** that takes a list of strings.
- This function will convert the data into a list of dictionaries.
- In this function create a list called **`errors`**.
- For each string in the list:
  - split the string on the commas
  - assign the components from the split to a dictionary with the following sequence of keys:
    1. name
    2. dob
    3. occupation
    4. gender
    5. married (cast to a bool)
    6. children (cast to an int)
- After processing the list of strings, return the list of dictionaries.

Example output:

```
In [23]: #HIDE

from pprint import pprint

def parse_data(data):
    fields = ('name', 'dob', 'occupation', 'gender', 'married', 'children')
    output = []
    errors = []
    for string in data:
        error = False
        row = {}
        values = string.split(',')
        for index, key in enumerate(fields):
            try:
                value = values[index].strip()
            except:
                errors.append(string)
                error = True
                break
            try:
                if key == 'married':
                    value = bool(value)
                elif key == 'children':
                    value = int(value)
            except:
                value = None
            row[key] = value
        if not error:
            output.append(row)
    print('Errors:')
    pprint(errors)
    print('-------')
    return output
```

```
In [21]: data = [
    "John Doe, 1972-03-28, carpenter, Male, False, 0",
    "Jane Doe, 1983-01-16, doctor, Female, True, 3"
]
```

```
parse_data(data)
```

```
Out[21]: [{'children': 0,
           'dob': '1972-03-28',
           'gender': 'Male',
           'married': True,
           'name': 'John Doe',
           'occupation': 'carpenter'},
          {'children': 3,
           'dob': '1983-01-16',
           'gender': 'Female',
           'married': True,
           'name': 'Jane Doe',
           'occupation': 'doctor'}]
```

Task 5b:

- Now we will try some error handling using **`try`** and **`except`** blocks.
- Assume that you will get some dirty data and add some try and except blocks in **`parse_data`** to handle it.
- In particular handle the following scenarios:
  - if the cast to `int` fails for `children` use `None`.
  - if the string is missing values (i.e. it has only 5 values instead of six) add the string to a list called **`errors`**, and skip over it.
  - Print out errors before returning the final list.

Example output:

```
In [24]: data = [
             "John Doe, 1972-03-28, carpenter, Male, False, 0",
             "Jane Doe, 1983-01-16, doctor, Female, True, 3",
             "John Deer, 1985-02-26, pilot, Male, Never, 0",
             "Jane Buck, 1987-11-06, lawyer, Female, True, five",
             "Jill Fawn, 1999-10-01, student, Female"
         ]

         parse_data(data)
```

```
Errors:
['Jill Fawn, 1999-10-01, student, Female']
-------
```

```
Out[24]: [{'children': 0,
           'dob': '1972-03-28',
           'gender': 'Male',
           'married': True,
           'name': 'John Doe',
           'occupation': 'carpenter'},
          {'children': 3,
           'dob': '1983-01-16',
           'gender': 'Female',
           'married': True,
           'name': 'Jane Doe',
           'occupation': 'doctor'},
          {'children': 0,
           'dob': '1985-02-26',
           'gender': 'Male',
           'married': True,
           'name': 'John Deer',
           'occupation': 'pilot'},
          {'children': None,
           'dob': '1987-11-06',
           'gender': 'Female',
           'married': True,
           'name': 'Jane Buck',
           'occupation': 'lawyer'}]
```