Introduction to Programming Homework 9

Due Monday Nov 28

You will turn in your homework **via GitHub**! Please use this link to start your repository :

https://classroom.github.com/assignment-invitations/6ceef9bb682bf468b1244c0310a37c52

Exercise 1 (Iterators)

Write a module called `iter_fun.py`

- **a.** Read about the itertools module at https://docs.python.org/3.5/library/itertools.html
- **b.** Write a **one line** function to count how many odd numbers are in a list. By a one line function I mean one of the following :

```
def count_odds(some_list) :
    return # your code
# or, if you prefer :
count_odds = lambda some_list : # your code
```

Remark : you do not need to do any input validation for all of these exercises.

- **c.** Write a **one line** function called `sum_upto_even(some_list)` to sum all the elements in a list up to but not including the first even number.
- **d.** Write a **one line** function called `triangle_num(n)` which returns a list of the first `n` triangle numbers starting with `0`.
- **e.** Given two lists `a,b` write a **one line** function `interleave(a,b)` which will return the list `[a[0], b[0], a[1], b[1], ...]`.
- **f.** Given a list of lists `data` write a **one line** function `interleave(*data)` which will return the list

```
[data[0][0], data[1][0], ..., data[k][0],
data[0][1], data[1][1], ..., data[k][1],
data[0][2], ...]
```

  where `k = len(data)-1`.
- **g.** Write a **one/two line** function `array_idx(n,d)` which **prints** all tuples $(i_0, i_1, \ldots, i_{d-1})$ with $0 \leq i_j < n$ for $0 \leq j < d$
- **h.** Write a **one/two line** function `array_upper(n,d)` which **prints** all tuples $(i_0, i_1, \ldots, i_{d-1})$ with $0 \leq i_0 \leq i_1 \leq \ldots i_{d-1} < n$.
- **i.** For a list A of numbers, the inversion count is
$$\mathrm{inv}(A) = \#\{(i,j) \mid i < j \text{ and } A[i] > A[j]\}.$$
  Write a **one line** function `inversion_count(A)` which returns $\mathrm{inv}(A)$.
- **j.** Write a **one line** function `total_inversions(n)` which returns $\sum_{p \in S_n} \mathrm{inv}(p)$ where S_n is the group of permutations of n elements.

- **k.** Write a **three/four line** function `print_groupby(data, group_key)` which takes a list `data` of dictionaries and prints the grouping based on the key `group_key`. For example, if

  ```
  data = [
      {'address': '5432 N CLACK', 'date': '09/01/2015'},
      {'address': '5118 N CLACK', 'date': '09/04/2015'},
      {'address': '5820 E TURNS', 'date': '09/02/2015'},
      {'address': '2232 N CLACK', 'date': '09/03/2015'},
      {'address': '5645 N REVINSDOON', 'date': '09/02/2015'},
      {'address': '1260 W ADRIZON', 'date': '09/02/2015'},
      {'address': '4331 N BRAIDWALL', 'date': '09/01/2015'},
      {'address': '1139 W GRANDVILLE', 'date': '09/04/2015'},
  ]

  group_key = 'date'
  ```

you should output

```
09/01/2015
    {'date': '09/01/2015', 'address': '5432 N CLACK'}
    {'date': '09/01/2015', 'address': '4331 N BRAIDWALL'}
09/02/2015
    {'date': '09/02/2015', 'address': '5820 E TURNS'}
    {'date': '09/02/2015', 'address': '5645 N REVINSDOON'}
    {'date': '09/02/2015', 'address': '1260 W ADRIZON'}
09/03/2015
    {'date': '09/03/2015', 'address': '2232 N CLACK'}
09/04/2015
    {'date': '09/04/2015', 'address': '5118 N CLACK'}
    {'date': '09/04/2015', 'address': '1139 W GRANDVILLE'}
```

where **each line is indented by 4 blank spaces.** The order in which the dictionary is printed is not important.

Exercise 2 (Callbacks)

Create a module called `file_watcher.py`.

The following code will watch for an update (save) to a file and call a callback function.

```python
import os
import time

noop = lambda *args, **kwargs: None

def watch_for_modify(path_to_file, max_time = 0., callback = noop) :
    try :
        start_time = time.monotonic() # to time our loop
        mod_time = None
        while time.monotonic() < start + max_time :
            # get the last save time for the file
            new_mod_time = os.path.getmtime(path_to_file)
            if new_mod_time != mod_time :
                callback(path_to_file)
                mod_time = new_mod_time
            time.sleep(1) # pause the while loop for 1 sec
    except :
        pass
```

Warning This watch code is rather crude and should never be used in the real world. I wanted to have something that works on all operating systems and is easy to understand.

- **a.** Learn about the `difflib` module and `difflib.unified_diff` at https://pymotw.com/3/difflib/#other-output-formats
- **b.** Create a class called `RunningDiff` which will have an instance method `print_diff(self, path_to_file)` such that calling

  ```python
  path_to_file = 'my_file.txt'
  diff = RunningDiff()
  watch_for_modify(path_to_file, max_time = 30., diff.print_diff)
  ```

  will print the changes to `'my_file.txt'` as you update and save it in a text editor.
  - you should create `'my_file.txt'` ahead of time
  - you should only print changes to `'my_file.txt'` and not the whole file in the beginning
  - to print the changes, just use `difflib.unified_diff` and print only the lines that were added or deleted