a4

ISTA 131, Fall 2019
Homework Assignment 4
Due: Thursday, 10/3/19 at 23:59 MST
Version 1.0

ttle less typing

Over time in my career I've found myself moving toward simple-as-possible names for things. It's time to apply that same practice here. Instead of putting your solution for this assignment in a file named `hw4.py`, name the file `a4.py`. Similarly, instead of the tests being in `hw4_test.py`, they are in `t4.py`. I'll refer to this assignment as `a4`, not `Hw4`. (I feel better already!)

tructions

Create a source file named `a4.py` containing definitions for the functions described below. When done, submit `a4.py` to the D2L dropbox for this assignment. It's fine to submit multiple versions; we'll grade the last one you submit before the deadline.

n't overlook the free-response questions!

This assignment includes several free-response questions. As of press time, `t4.py` doesn't have any awareness of them, and thus won't remind you to do them. **Don't forget them!**

Time permitting, we're happy to give feedback on answers to questions. Mail to `ista131-questions`.

n't Post Your Code on Piazza!

Remember: If you have a question about your code, DO NOT post it on Piazza! Instead, mail it to `ista131-questions`.

It's fine to post Piazza questions that include code from the unit tests.

If a Piazza question you post seems to have mysteriously disappeared, DO NOT REPOST IT! Instead, mail to `ista131-questions`. It may be the case that the question accidentally revealed part of a solution and was deleted by an SL or me and we've yet to mail you about it.

If undecided about posting on Piazza or using mail, use mail!

ring

Point values are shown for each problem. There are a total of 100 points of problems.

Attention to Restrictions

The specification for some problems may include restrictions, like not being able to use a built-in function or have certain characters in your source code. Just like many exercises in the gym focus on various muscle groups, I use restrictions to help you build certain programming "muscles". If your code violates a restriction, you'll get no points for that problem.

Restrictions will always be marked with "**RESTRICTION**", so that you can easily use your browser to search for restrictions.

Time permitting, we're happy to inspect your code before it's due to see if there are any violations of restrictions. Just mail it to `ista131-questions`.

ting

The zip you download from D2L will include a unit testing module, `t4.py`.

The lecture 2 slides have a section that talks about the unit testing machinery in some detail.

Remember that I strongly recommend testing "by hand" before you try the unit test for a problem! Use "`python -i a4.py`" to load up your code and put you into the REPL so that you can experiment with it one expression at a time.

Many times a student has come to office hours with a complex test failure they don't understand, only to see that a quick test done by hand shows that their code doesn't work with even a very simple case.

When testing by hand has given you confidence that your code is working at least for simple cases, then, and only then, should you try the unit tests for the problem at hand.

In brief, to test only one function at a time, the practice I recommend when developing, use the `-k` option, which specifies to only run tests with names that contain the given string. Example:

```
python -m unittest tN.py -k somestring
```

To test all the functions just leave off the `-k` and its argument. Example:

```
python -m unittest tN.py
```

Adding the `-v` (verbose) option will show which tests are being run. Example:

```
python -m unittest tN.py -v
```

The unit tests (in `tN.py`) are considered part of the spec; they provide examples of usage of the functions. We may use additional tests when grading but if you pass all supplied tests, your score will likely be no worse than 85 out of 100.

:umentation

No comments whatsoever are required. Comment as you see fit.

ıding

Your code will be graded on correctness and good use of Python. Even if your code passes all tests, we may ask you to rewrite code that is not clear, concise, and idiomatic, and/or it shows evidence of fundamental misunderstandings.

cheating!

If you find yourself tempted to cheat, take another look at slides 17-21 in the first lecture to remind yourself of the severe consequences of being caught.

Remember: You've got six people ready to help you with this assignment! Take advantage of office hours, email, Piazza, and IM. Never hesitate to ask for help—that's what you're paying us for! And don't wait to get started on this assignment!

---

blem specifications

- **`sums`**: (8 points) This function takes an `ndarray` and returns a 3-tuple where there the first element is an array with the sum of each of the array's rows, the second element is an array with the sum of each of the array's columns, and the third is simply the sum of all the array's elements.

  RESTRICTION: Your solution may not contain any loops or (if you know what the term "recursive" means) be "recursive". Let NumPy do the looping instead.

  Example:

```
>>> a = np.random.randint(1,10,size=(3,5))
>>> a
array([[8, 3, 6, 7, 4],
       [9, 8, 3, 5, 1],
       [8, 7, 3, 1, 1]])
>>> sums(a)
(array([28, 26, 20]), array([25, 18, 12, 13
```

- **`fvr`**: (12 points) For this problem you are to write a function that behaves like `np.ndarray.flatten`, but does the flattening by using `np.ndarray.reshape`. (The name "`fvr`" stands for "flatten via reshape".)

  RESTRICTION: Needless to say (I hope!), you can't use `np.ndarray.flatten` in your solution for `fvr`!

  I recommend that you start this problem by first looking at the example of `flatten` on slide 5.23. Then experiment with flattening arrays of varying shapes. Then, do `help(np.ndarray.flatten)`. You'll see that `flatten` has a default parameter named `order`. Similarly, `fvr` has a

default `order` parameter, too.

Examples: (some from the built-in documentation)

```
>>> a = np.array([[1,2], [3,4]])
>>> a.flatten()
array([1, 2, 3, 4])

>>> fvr(a)
array([1, 2, 3, 4])

>>> a.flatten(order='F')
array([1, 3, 2, 4])

>>> fvr(a, order='F')
array([1, 3, 2, 4])
```

If `order` is specified, assume it will be `'C'` or `'F'` in either upper- or lower-case, and in particular, you do not need to support the `'A'` or `'K'` orderings.

You might find the `T` attribute of `ndarray` to be useful. It produces a "transpose" of the array:

```
>>> a
array([[1, 2],
       [3, 4]])

>>> a.T
array([[1, 3],
       [2, 4]])
```

- **`znm`**: (16 points) This function takes an `ndarray` and returns a new array that is a copy of its argument, but with all elements of each row set to zero, except for the first occurrence of the largest value in each row. There is an optional second argument, named `rows` that is assumed to be a `bool`. If it is `False`, then the operation is applied to columns instead of rows.

Example:

Example:

```
>>> r
array([[3, 2, 5, 8, 9],
       [1, 4, 8, 5, 8],
       [4, 5, 5, 9, 5],
       [4, 5, 4, 4, 8]])

>>> znm(r)
array([[0, 0, 0, 0, 9],
       [0, 0, 8, 0, 0],
       [0, 0, 0, 9, 0],
       [0, 0, 0, 0, 8]])

>>> znm(r,rows=False)
array([[0, 0, 0, 0, 9],
       [0, 0, 8, 0, 0],
       [4, 5, 0, 9, 0],
       [0, 0, 0, 0, 0]])
```

Use `ndarray.copy` to make a copy of the array. You might find `ndarray.argmax` to be useful, too.

- **`check_sdk`**: (24 points) During lecture we worked on a partial Sudoku checker. For this problem you're to develop a full checker, handling rows, columns, and squares.

  `check_sdk` takes one argument, a 9x9 array of integers that represents a Sudoku board and checks to see if each row, column, and 3x3 square (non-overlapping) contain each of the digits from 1-9. If that's true, `check_sdk` prints `"OK!"`. If not, the errors are enumerated, again by printing.

  Here's a correct board:

```
>>> print(b1)

      5 3 4 | 6 7 8 | 9 1 2
      6 7 2 | 1 9 5 | 3 4 8
```

```
1 9 8 | 3 4 2 | 5 6 7
- - - + - - - + - - -
8 5 9 | 7 6 1 | 4 2 3
4 2 6 | 8 5 3 | 7 9 1
7 1 3 | 9 2 4 | 8 5 6
- - - + - - - + - - -
9 6 1 | 5 3 7 | 2 8 4
2 8 7 | 4 1 9 | 6 3 5
3 4 5 | 2 8 6 | 1 7 9
```

```
>>> type(b1)
<class 'str'>

>>> check_sdk(make_board(b1))
OK!
```

Here's a board with a single digit that's wrong:

```
>>> print(b2)

    5 3 4 | 6 7 8 | 9 1 2
    6 7 2 | 1 9 5 | 3 4 8
    1 9 8 | 3 4 2 | 5 6 7
    - - - + - - - + - - -
    8 5 9 | 7 6 1 | 4 2 3
    4 2 6 | 8 5 3 | 7 9 1
    7 1 3 | 9 2 4 | 8 5 6
    - - - + - - - + - - -
    8 6 1 | 5 3 7 | 2 8 4
    2 8 7 | 4 1 9 | 6 3 5
    3 4 5 | 2 8 6 | 1 7 9

>>> type(b2)
<class 'str'>

>>> check_sdk(make_board(b2))
```

```
>>> check_sdk(make_board(b2))
Bad row(s): 7
Bad column(s): 1
Bad square(s): (7, 1)
```

Notice that the errors report the row and column numbers, and the coordinates of the upper left-hand corner of squares, using one-based numbering, just like non-programmers do!

Here's a board with several mistakes:

```
>>> print(b3)

      5 3 4 | 6 7 8 | 9 1 2
      6 7 2 | 1 9 5 | 3 4 8
      1 9 8 | 3 4 2 | 5 6 7
      - - - + - - - + - - -
      8 5 9 | 7 6 1 | 4 2 3
      4 2 6 | 8 5 3 | 7 9 1
      7 1 3 | 9 2 4 | 8 5 6
      - - - + - - - + - - -
      9 6 1 | 9 4 3 | 3 8 4
      2 8 7 | 1 2 2 | 6 3 5
      3 4 5 | 2 1 2 | 1 7 9

>>> type(b3)
<class 'str'>

>>> check_sdk(make_board(b3))
Bad row(s): 7 8 9
Bad column(s): 4 5 6 7
Bad square(s): (7, 4) (7, 7)
```

Implementation notes:

- The zip you download from D2L will include `make_board.py`, which has code for the `make_board` function. Copy the code for `make_board` into your `a4.py`.

make_board into your a4.py!

- My `check_sdk` function uses three helper functions: `check_rows`, `check_squares`, and `check_19`. `check_squares` is very much like the partial checker we worked on in class. `check_19` is a boolean function that returns `True` iff an array contains exactly the integers from one through nine. I was about to write a `check_cols` function, too, but then I realized that `check_rows` could be called twice, once to check rows and then again to check columns. (Think about it!)

- I really should have cast this problem as a `Board` class with a `check` method but that didn't cross my mind until it was too late. However, it would surely be good practice for the first mid-term to do that on your own.

- **`concentrate`**: (20 points) This function takes a positive odd integer N and a NumPy array `a` whose width and height are both a multiple of the integer N. For each NxN block of values in `a`, `concentrate` replaces the center value with the sum of the values in the NxN block and then zeroes the other values. It returns `a`.

  My initial "picture" for this problem was sweeping all data values for some region into a pile in the center of the region.

  Examples:

```
>>> c0
array([[9, 1, 6],
       [9, 3, 5],
       [6, 2, 5]])

>>> c0.sum()
46

>>> concentrate(c0,3)
array([[ 0,  0,  0],
       [ 0, 46,  0],
       [ 0,  0,  0]])

>>> c0
```

```
>>> c0
array([[ 0,  0,  0],
       [ 0, 46,  0],
       [ 0,  0,  0]])

>>> c1
array([[1, 2, 0, 1, 0, 2, 0, 1, 1],
       [0, 1, 1, 1, 0, 2, 2, 2, 1],
       [0, 2, 2, 0, 2, 2, 2, 0, 1],
       [1, 0, 0, 2, 1, 2, 2, 2, 2],
       [1, 1, 2, 1, 1, 2, 1, 0, 2],
       [2, 1, 0, 2, 0, 0, 0, 0, 1]])

>>> concentrate(c1,3)
array([[ 0,  0,  0,  0,  0,  0,  0,  0,  0]
       [ 0,  9,  0,  0, 10,  0,  0, 10,  0]
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0]
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0]
       [ 0,  8,  0,  0, 11,  0,  0, 10,  0]
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0]

>>> c2
array([[2, 1, 2, 2, 1, 1, 2, 1, 1, 0],
       [2, 2, 1, 2, 0, 0, 0, 1, 0, 1],
       [2, 2, 2, 1, 0, 2, 2, 0, 2, 1],
       [2, 1, 2, 0, 1, 2, 0, 1, 2, 1],
       [0, 0, 2, 2, 2, 1, 1, 0, 2, 0]])

>>> concentrate(c2,5)
array([[ 0,  0,  0,  0,  0,  0,  0,  0,  0,
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,
       [ 0,  0, 34,  0,  0,  0,  0, 24,  0,
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,
```

Free-response questions

Free-response questions

The following problems are free-response questions to be answered in English, just like the free-response questions on homework assignment 3. In some cases your answer is required to include a transcript of interaction with the Python REPL.

- Question 1: (4 points) A friend would like `ndarray.reshape` to figure out how many columns are needed given that she'd like N values to be formed into R rows. She speculates that perhaps just providing one argument to `reshape` might cause the other to default, but it doesn't. For example, she tried to get three rows of four columns like this:

```
>>> np.array(range(12)).reshape(3)
ValueError: cannot reshape array of size
```

  What could she do to get the number of rows automatically calculated? Be sure to provide an example of interaction for her!

- Question 2: (4 points) Fill in the blank! There is an `ndarray` method that behaves like this:

```
>>> a = np.array(range(10)).__________()
>>> a
array([ 0,  1,  3,  6, 10, 15, 21, 28, 3
```

  What's the name of the method? Show two interesting examples of using it.

  I sure hope I don't hear anybody giving away the name of the method!

- Question 3: (**8** points) For two points each, write each of the following lines of code in English. Your answer should include both the Python code, and its English "translation".

```
a=np.arange(100,125).reshape(5,5)

a += 5

print(a[-3: 1:1)
```

```
print(a[:3,:4])

a[:,a.shape[1]//2] = a.sum() # assume a
```

- Question 4: (4 points) You guys really dazzled me with the questions you wrote for question 5 on assignment 3! I was so impressed that I even posted about it on Facebook.

  I'm curious to see if you can dazzle me again, so for four points write a question related to NumPy. Follow the style of the questions above, and express your question as a triple-quoted string literal returned by the function `q4`.

Extra Credit: Hours and Observations

- I always want to have some idea of how much time students are spending on assignments. For two points of extra credit include a function named `estimated_hours` that simply returns your best estimate of how many hours you spent working on this assignment. Example:

  ```
  def estimated_hours():
      return 9.5  # must be an int or a fl
  ```

  There's no ability to add any comment to it, like "I did this while watching TV", so if you spent ten hours on it in front of the TV, you might then decide that translates to four or five hours of undistracted work, but you've got to make a guess at that factor for you.

- For up to ten points of extra credit, cite an interesting course-related observation (or observations) that you made while working on the assignment. The observation should have at least a little bit of depth. Feedback and comments about the assignment are welcome, too. Was it too long, too hard, too detailed? Speak up! I appreciate all feedback, favorable or not.

  Communicate those observations to us by including a function named `observations` that returns a string with your observations, feedback, whatever. You can use a triple-quoted string to preserve formatting. See the homework 1 write-up for a real example but here's an abstract one:

  ```
  def observations():
      """
  ```

```
    return """
...observation 1...

...observation 2...
"""
```

Feel free to write as much you want but it's quality, not quantity, that will