Quiz 3 Conflict

The questions below are due on Wednesday May 09, 2018; 06:55:00 PM.

6.009 Quiz 3 Conflict -- Spring 2018

Please download the .zip file below, which contains all the files, test cases, etc., you need to complete the quiz using your own machine. When unzipped it will create its own d
files. The instructions follow below. You'll be editing `quiz.py` to add the code requested by the instructions.

Download: quiz3_conflict.zip (https://eecs6009.mit.edu/spring18/quizzes/quiz3_conflict/quiz3_conflict.zip)

Submission

When you have tested your code sufficiently on your own machine, submit your modified `quiz.py` below by clicking on "Choose File", then clicking the `Submit` button to send
server. To receive credit you must upload your code *before* the timer expires.

The server will run the tests and report back the results below but be aware that the server may be backlogged at the end of the quiz session. Once the server has indicated th
queued to be checked, you're all set -- your submission has been accepted as on time and you don't need to wait for the results to be reported.

Download Your Last Submission (https://eecs6009.mit.edu/cs_util/get_upload?
path=%5B%22spring18%22%2C+%22quizzes%22%2C+%22quiz3_conflict%22%5D&fname=cyjchoi___q000000___1525906256.261618___bbc283127490a650
Click to View Your Last Submission (https://eecs6009.mit.edu/spring18/lab_viewer?lab=quizzes/quiz3_conflict&name=q000000&as=cyjchoi)

Select File `No file selected`

Submit

You have submitted this assignment 2 times.

Instructions

This quiz assumes you have Python 3.5 (or a later version) installed on your machine.

This quiz is **closed-book and closed-Internet** -- you are only allowed to access the quiz page on the 6.009 website and the material provided in the .zip file. You are not all
information on your laptop or on the web. **You may not discuss the quiz with other students -- even after you complete the quiz and/or complete any regrade
regrades have been completed for all students and final quiz grades assigned (at the end of next week),** because many students will still be working on regrades
the quiz.

When you unzip the .zip file, you'll find `library.pdf`, which includes the chapters 1-4 of the Python Library Reference -- they cover the built-in functions (Chap. 2) and built-in
Note that Python's built-in `help` function will provide a concise summary of built-in operations on data types (e.g., `help(set)`) or information about the arguments for the buil
`help(sorted)`).

Proctors will be available to answer administrative questions and clarify specifications of coding problems, but they should not be relied on for coding help.

As in the labs, `test.py` can be used to test your code in `quiz.py`. Each of the three problems has a unit test: TestProblem1, TestProblem2, etc. Remember that you can run tes
problem like so

```
python3 test.py TestProblem1
```

This quiz is worth 20 points. Your quiz score will be based primarily on performance of your last code submitted during the quiz, tested against 20 test cases. However, points
other circumstances:

- You **must not not import any Python modules** to use as part of your solutions -- quizzes with `import` statements (or their equivalent!) will be given grades of 0.
- Having your code check for a specific input (called "hard coding") and then return a specific result is okay only for the base cases in iteration or recursion. Problem soluti
specific test case arguments other than base cases are disallowed and will receive no credit.
- You do not have to include doctests or detailed comments in your code: we will not deduct any points for lack of comments, but encourage those as general good practi

Aside from these special point deductions above, your quiz score will be based on the number of unittest tests your last submitted code passes. If you pass all of the tests, you

credit. If you pass 15 of the 20 tests, you'll receive 75% of the base credit for the quiz (i.e., 15 points). Note also that to pass some of the tests in the time allotted, your solutio reasonably efficient.

For your own debugging, you are welcome to add test cases of your own devising to `test.py` (but note that any tests you add will not be run on the server).

Problem 1. Run Length Encoding (5 points)

Run length encoding is a simple technique to compress sparse data sets featuring long runs (sequences of the same value). In this problem, we will take as input a 2-D array (lis an image, and we will compress it using this technique.

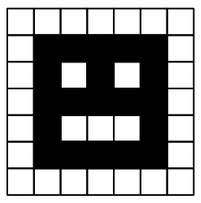
We will work our way from the top-left of the image, compressing sequences of n copies of the same number c into a single tuple (n, c).

For example, consider the following small 2-D array:

```
[[0, 0, 0],
 [2, 2, 1],
 [1, 1, 1]]
```

In compressed form, this would be a list `[(3, 0), (2, 2), (4, 1)]` (representing the fact that, reading from left-to-right and top-to-bottom, we first see a sequence of 3 `0`'s, `2`'s, and then a sequence of 4 `1`'s).

As another example, the following "smiley face" image:



is represented by this array:

```
[[1, 1, 1, 1, 1, 1, 1],
 [1, 0, 0, 0, 0, 0, 1],
 [1, 0, 1, 0, 1, 0, 1],
 [1, 0, 0, 0, 0, 0, 1],
 [1, 0, 1, 1, 1, 0, 1],
 [1, 0, 0, 0, 0, 0, 1],
 [1, 1, 1, 1, 1, 1, 1]]
```

In compressed form, this would be `[(8, 1), (5, 0), (2, 1), (1, 0), (1, 1), (1, 0), (1, 1), (1, 0), (2, 1), (5, 0), (2, 1), (1, 0), (3, 1), (1, 0), (2, 1), (5`

Complete the definition of the `run_length_encode_2d` function in `quiz.py` to implement the run-length encoding process. Your code should be able to handle any 2-D array o and zeros).

Problem 2. Word Squares (8 Points)

A *word square* is a puzzle that consists of a set of words written out in a square grid, such that the same words can be read both vertically and horizontally. For example, the fo square of length 3:

```
bit
ice
ten
```

Notice that both the first row and the first column spell out "bit," the second row and the second column each spell out "ice," and the third row and the third column spell out '

We will represent word squares as tuples of words representing the rows of the square. For example, the square above would be represented as the tuple `('bit', 'ice', 'te`

Write a *generator* `word_squares(top)` that takes a `top` word as argument (`'bit'` in the example above) and generates all valid word squares containing that top word. You are `allwords` that contains all words that should be considered valid English words (your word square should contain only valid words).

Note that in order to pass some of the tests, your implementation may need to be fast enough to complete that test within a 60-second timeout on the server.

For example:

```
>>> word_squares('no')
<generator object word_squares at SOME_MEMORY_LOCATION>
>>> set(word_squares('no'))
{('no', 'or'), ('no', 'oh'), ('no', 'of'), ('no', 'on'), ('no', 'ox')}
```

Problem 3. Fishing (7 Points)

In this problem, we will implement a simulation for a fishing game. In this game, we have several different types of fish, each of which has a preferred meal.

The game is broken into discrete timesteps (the game starts at time 0). Fish appear at certain locations at certain times, and can be caught during a particular time frame if you preferred foods as bait.

One example class structure for fish is given by `Fish` and its subclasses in `quiz.py`. Note that any instance of a subclass of `Fish` will have an `eats` attribute.

Each `Fish` instance has several instance attributes:

- `weight` is the weight of the fish (in pounds)
- `arrive_at` is an integer representing the time at which the fish arrives
- `duration` is an integer representing the number of timesteps for which the fish can be caught.

A class called `Pond` will represent our game. You are responsible for implementing the following methods in the `Pond` class:

- `add_fish(location, fish)` should add a new fish to the game at the given location. `location` is a `(row, column)` tuple, and `fish` is an instance of the `Fish` class (or a s should not increment the time step.
- `catch_fish(location, bait)` should attempt to catch a fish on the current timestep, and then it should increment the time step by one. `location` is a `(row, column)` tu string containing the bait to be used. If there is currently a fish at the given location that eats the given bait, then that fish instance should be returned (and that same fis again). If there is no fish currently at that location that eats the given bait, the method should return `None`. If there are multiple fish that could be caught, you should retu smallest weight (if there are multiple fish with the same smallest weight, you should return the one that was added to the pond first).
- `wait(n)` should increment the timestep by `n` (an integer).
- `weight_caught()` should return the total weight (in pounds) of all fish caught to this point. It should not increment the time step.

You are welcome to add additional attributes/methods to the `Pond` class, but your code should not rely on adding features to the `Fish` class.

Consider the following example, using the classes from `quiz.py`:

```
x = Pond()
x.add_fish((0,0), SmallmouthBass(7, 3, 10)) # 7 pounds, location (0,0), can be caught on timesteps 3 through 12 (inclusive)
x.catch_fish((0,0), 'insect')  # timestep 0.  Returns None (fish has not yet arrived).
x.add_fish((0,1), BubbleBass(10, 0, 5)) # 10 pounds, location (0,0), can be caught on timesteps 0 through 4 (inclusive)
x.weight_caught() # 0, since we haven't caught anything.
x.catch_fish((0,1), 'insect')  # timestep 1.  Returns None (wrong bait).
x.wait(3)
x.catch_fish((0,1), 'krabby patty')  # timestep 4.  Returns the fish (correct bait, fish is still around).
x.weight_caught() # 10, for the one fish we've caught.
x.catch_fish((0,0), 'insect')  # timestep 5.  Returns the fish (correct bait, fish is still around).
x.weight_caught() # 17, the sum of the weights of the two fish we've caught.
```