First assignment: Quizzes and Graphs

Per Starbäck

26th January 2020

Quizzes

Make a file `quiz.py` in which you define a class `Question`, meant for a question in a quiz, usable like this:

```
>>> q = Question('Capital of Sweden?', 'Stockholm')
>>> q.add_wrong('Uppsala')
>>> q.add_wrong('Oslo')
```

In attributes it should have the text of the question, the answer, and a list (or set) of wrong answers, initially empty.

A method `ask` should ask the question, and list all the possible answers numbered and prompt the user to answer by giving a number. Like this:

```
>>> q.ask()
Capital of Sweden?
1: Oslo
2: Uppsala
3: Stockholm
Which is your answer? 3
(3, 3)
>>> q.ask()
Capital of Sweden?
1: Stockholm
2: Uppsala
3: Oslo
Which is your answer? 2
(2, 1)
>>>
```

All the alternatives (the correct one, and the wrong answers) should be listed in a random order

The method should *return* two values (that is, a tuple with two elements) as shown in the example: The alternative the user answered, and the correct alternative, as integers.

Also create a class `Quiz`. It should have attributes `name` (used when creating it) and `questions` (initially an empty list). A question can be added to a quiz like this:

```
>>> quiz = Quiz('Example quiz')
>>> quiz.add_question(q)
```

You are given a function `create_quiz_from_file` which can read a file such as the supplied `disney.quiz`. A quiz should have a method '`do`' by which you do/take a quiz, and you should be able to get output similar to this:

```
>>> dquiz = create_quiz_from_file('disney.quiz')
>>> dquiz.do()
Disney films
============
Who is Olaf in Frozen?
1: A dragon
2: A prince
3: A reindeer
4: A snowman
Which is your answer? 2
Sorry, no. Correct answer: 4

Which film is 'When You Wish Upon a Star' from?
1: Snow White and the Seven Dwarfs
2: Pinocchio
3: Aladdin
4: Dumbo
Which is your answer? 2
Correct!

Who did the voice for king Mufasa in The Lion King?
1: Ian McKellen
2: Christopher Lee
3: James Earl Jones
4: David Ackroyd
Which is your answer? 1
Sorry, no. Correct answer: 3

You answered 1 of 3 correctly.
>>>
```

(The comments about correct and incorrect answers don't have to be exactly as in the example, but should contain the same information.)

The point of `create_quiz_from_file` is to make a somewhat simple way to write a quiz by hand. The function as given is not very fool-proof, though. It will yield errors or give unexpected results if the input isn't as it should.

Also let a quiz have a method `do_until_right` which makes you do the quiz over and over again until you answer everything correctly.

Integer questions

Above are only *multiple-choice* questions. Also make a class `IntQuestion` for a question which has an *integer* as the correct answer. No alternatives are listed when those questions are asked, so the user has to answer the number correctly without any help. Let this class inherit from `Question` and only change what is needed.

So the same quiz could have a mixture of questions of different kinds in it. Update `create_quiz_from_file` so that it can accept an integer question of this form in a `quiz` file:

```
iq What year is the movie Bolt from?
a 2008
```

When the above is working you have a fine first version! You can submit it before you continue under the name `quiz1.py` to get feedback if you want, so you know you are on the right track. The filename is important, since that's how we know it's not your real submission (= look at it later), but something for early feedback.

Choosing alternatives

Next, for multiple-choice questions list at most *four* answers to choose from when asking the question, even if there are more. Actually define a constant[1] with the value 4 for this, so that the program can be changed to have some other number by just changing one line.

When choosing which wrong answers to show we will do it in a way that makes the quiz harder, by displaying wrong alternatives that the user has chosen more often before. So if any alternatives are obviously wrong they will be avoided, and the correct answer will be presented together with the three *best* of the wrong alternatives.

Because of that a wrong answer should no longer be just a string, but an *object* that also remembers how many times that answer has been offered as an alternative and how many times the user has answered that.

That's a new class for you to define. The `add_wrong` method used above should still take a string as argument, but now create such an object.

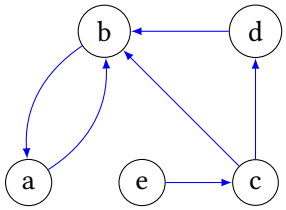
When choosing the three wrong answers to display (if there are more than that available) pick those with the highest score s according to the formula

$$s = \frac{2c + 1}{d + 1} \tag{1}$$

where c is the number of times that alternative has been chosen and d is the number of times it has been displayed. You may have use for the Python HOW TO on sorting.[2]

Equation 1 has single-letter s, c and d, because that's how you usually do in math. As you understand the attributes for the new class you define should have longer names than that!

Directed Graphs



A *directed graph* (or *digraph*) is made up of a set of *vertices* connected by *edges*, where the edges have a direction.

It is defined as an ordered pair $G = (V, A)$ where

- V is a set of elements, the vertices,
- A is a set of ordered pairs of vertices, called arrows or edges

You should make a file `digraph.py` which contains a class `Digraph` for digraphs.

This *could* be stored simply as two lists in Python, a list of vertices and a list of pairs (probably tuples). But you are free to represent it in some other way, and you should *encapsulate* the internal representation and implementation anyway, so you use this class without having to know or care about how it is represented inside.

(Since the mathematical definition talks about *sets*, maybe it would be useful to take a look at the Python

[1] https://www.python.org/dev/peps/pep-0008/#constants

[2] https://docs.python.org/3/howto/sorting.html

datatype `set`, even though we never used that in Programming 1.)

There should be methods to get the vertices and the edges. So if `d` is a digraph then `d.vertices()` should return the vertices and `d.edges()` return the edges. These return values should be lists or something else it's possible to loop over, and the edges should be tuples of length 2 (or something sufficiently similar), so code using it could do for example

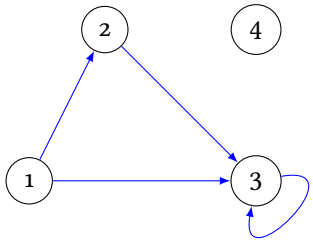
```
for source, dest in d.edges():
    if source == dest:
        print(f'Detected loop from/to {source}')
```

to find all *loops* (edges going from and to the same vertex) in a digraph, like that to/from vertex 3 in the next figure.

Also it should be possible to instantiate a digraph giving the edges and the vertices (by name). So

```
d = Digraph(edges=[(1,2), (1,3), (2,3), (3,3)],
            vertices=[1,2,3,4])
```

should result in the graph below.



Actually it seems a bit unnecessary to list the vertices that are given in the edge information anyway. So for convenience make it so that all vertices used by any edge are added automatically, and the parameter `vertices` only needs to mention *additional* (isolated) vertices. So a similar graph should be returned by

```
Digraph(edges=[(1,2), (1,3), (2,3), (3,3)],
        vertices=[4])
```

and just

```
Digraph(edges=[(1,2), (1,3), (2,3), (3,3)])
```

gives a smaller graph that doesn't have an isolated node.

The default for `edges` should be no edges, so just `Digraph()` should return a digraph with no vertices and no edges.

Modifying digraphs

Make a Digraph method `.add_edge(source, dest)` which adds an edge to a digraph (also adding a vertex if needed).

Make a Digraph method `.remove_vertex(vertex)` which removes a vertex and all edges from/to that vertex.

Transitivity

A relation R is *transitive* if for all elements $a, b, c \in A$, whenever R relates a to b and b to c, then R also relates a to c.

For a Digraph that transitivity means that whenever there is an edge from a to b and an edge from b to c there is also an edge from a to c.

Make a Digraph method `.is_transitive` which returns a boolean value being true if and only if the digraph is transitive!

(Doing that in an efficient way for large graphs can be hard, but here it's quite enough with something that follows the definition more or less directly and works fine for small graphs.)