UDACITY

PROJECT

Tournament Results

A part of the Full Stack Web Developer Nanodegree Program

PROJECT REVIEW

CODE REVIEW 12

NOTES

vagrant/tournament/tournament.py 7

```
#!/usr/bin/env python
#
# tournament.py -- implementation of a Swiss-system tournament
#

import bleach
import psycopg2


def run_query(statement, params=None):
    '''Helper method to run queries'''
    conn = connect()
    cur = conn.cursor()
    cur.execute(statement, params)
    conn.commit()
    conn.close()


def run_query_one(statement, params=None):
    '''Helper method to run queries that return one result'''
    conn = connect()
    cur = conn.cursor()
    cur.execute(statement, params)
    result = cur.fetchone()[0]
    conn.close()
    return result


def run_query_args(statement, params=None):
    '''Helper function for queries that return multiple columns'''
    conn = connect()
    cur = conn.cursor()
    cur.execute(statement, params)
    results = cur.fetchall()
    conn.close()
    return results

```

AWESOME

Excellent job implementing these utility methods to avoid code repetition.

```
def connect():
    """Connect to the PostgreSQL database.  Returns a database connection."""
    return psycopg2.connect("dbname=tournament")
```

SUGGESTION

You can refactor your connect() method to deal not only with the database connection but also with the cursor since you can assign and return multiple variables simultaneously.

In the stage of setting up the connection with the database, sometimes you may encounter different exceptions. In practice, this crucial stage should be handled very carefully by using the try/except block similar to the code below.

e.g.

```
def connect(database_name="tournament"):
    try:
        db = psycopg2.connect("dbname={}".format(database_name))
        cursor = db.cursor()
        return db, cursor
    except:
        print("<error message>")

def registerPlayer(name):
    db, cursor = connect()

    query = "INSERT INTO players (name) VALUES (%s);"
    parameter = (name,)
    cursor.execute(query, parameter)

    db.commit()
    db.close()
```

```
42
43
44 def deleteMatches():
45     """Remove all the match records from the database."""
46     run_query("DELETE FROM match")
47
48
49 def deletePlayers():
50     """Remove all the player records from the database."""
51     run_query("DELETE FROM players")
```

SUGGESTION

You can also use `TRUNCATE` here. That will be much faster operation than `DELETE`. In case of tables with foreign key relationships, use `CASCADE`. You can read more about it on the link provided below.

https://www.postgresql.org/docs/9.1/static/sql-truncate.html

```
52
53
54 def countPlayers():
55     """Returns the number of players currently registered."""
56     return run_query_one("SELECT COUNT(*) FROM players")
57
58
59 def registerPlayer(name):
60     """Adds a player to the tournament database.
61
62     The database assigns a unique serial id number for the player.  (This
63     should be handled by your SQL database schema, not in your Python code.)
64
65     Args:
66       name: the player's full name (need not be unique).
67     """
68     scrubbed_name = bleach.clean(name)
```

SUGGESTION

Although its great that you have used `bleach` to sanitize the input, it's not necessary for the current project because the inputs are from within the system. Usually, bleach is required when you are dealing with data coming in from external sources for example HTML forms.

```
69     run_query("INSERT INTO players (name) VALUES (%s)", (scrubbed_name, ))
```

AWESOME

Great job using query parameters to prevent SQL injection attacks.

```
70
71
72 def playerStandings():
73     """Returns a list of the players and their win records, sorted by wins.
74
75     The first entry in the list should be the player in first place, or a
76     player tied for first place if there is currently a tie.
77
78     Returns:
79       A list of tuples, each of which contains (id, name, wins, matches):
80         id: the player's unique id (assigned by the database)
81         name: the player's full name (as registered)
82         wins: the number of matches the player has won
```

```
83          matches: the number of matches the player has played
84      """
85      return run_query_args(
86          "SELECT * FROM standings ORDER BY wins DESC, matches ASC")
```

SUGGESTION

You can simplify this SQL statement further by moving `ORDER BY` clause into the view.

```
87
88
89  def reportMatch(winner, loser):
90      """Records the outcome of a single match between two players.
91
92      Args:
93        winner:  the id number of the player who won
94        loser:  the id number of the player who lost
95      """
96      run_query("INSERT INTO match(winner, loser) VALUES (%s, %s)",
97                (winner, loser))
98
99
100 def swissPairings():
101     """Returns a list of pairs of players for the next round of a match.
102
103     Assuming that there are an even number of players registered, each player
104     appears exactly once in the pairings.  Each player is paired with another
105     player with an equal or nearly-equal win record, that is, a player adjacent
106     to him or her in the standings.
107
108     Returns:
109       A list of tuples, each of which contains (id1, name1, id2, name2)
110         id1: the first player's unique id
111         name1: the first player's name
112         id2: the second player's unique id
113         name2: the second player's name
114     """
115     standings = playerStandings()
```

AWESOME

Great Job reusing `playerStandings()` method

```
116     total_players = len(standings)
117     pairings = []
118
119     for player in range(0, total_players, 2):
120         pair = ((standings[player][0], standings[player][1],
121                  standings[player + 1][0], standings[player + 1][1]))
122         pairings.append(pair)
123
124     return pairings
125
```

▸ vagrant/tournament/tournament.sql 5

▸ vagrant/tournament/tournament_test.py

▸ vagrant/forum/solution/forumdb_steptwo.py

▸ vagrant/forum/solution/forumdb_stepone.py

▸ vagrant/forum/solution/forumdb_solved.py

▸ vagrant/forum/solution/forumdb_initial.py

▸ vagrant/forum/solution/forum.py

▸ vagrant/forum/forumdb.py

▸ vagrant/forum/forum.sql

▸ vagrant/forum/forum.py

▸ vagrant/catalog/README.txt

README.md

RETURN TO PATH

Rate this review

Student FAQ