UDACITY

PROJECT

Logs Analysis

A part of the Full Stack Web Developer Nanodegree Program

PROJECT REVIEW

CODE REVIEW 4

NOTES

▾ reporting.py 4

```
1  #!/usr/bin/env python3
```

AWESOME

shebang, shebang!

```
2  import psycopg2
3
4  DB_NAME = 'news'
5  DB_USER = 'vagrant'
6
7
8  def get_most_popular_articles():
9      """gets the most popular three articles of all time,
10     as a sorted list with the most popular article at the top."""
11     # Connect to an existing database
12     conn = psycopg2.connect("dbname=%s user=%s" % (DB_NAME, DB_USER))
```

SUGGESTION

What if something goes wrong with the database connection? The connect method will raise an Exception (http://initd.org/psycopg/docs/module.html#exceptions). I recommend you refactor your database connection code into the following:

```
def connect(database_name):
    """Connect to the PostgreSQL database.  Returns a database connection."""
    try:
        db = psycopg2.connect("dbname={}".format(database_name))
        c = db.cursor()
        return db, c
    except psycopg2.Error as e:
        print "Unable to connect to database"
        # THEN perhaps exit the program
        sys.exit(1) # The easier method
        # OR perhaps throw an error
        raise e
        # If you choose to raise an exception,
        # It will need to be caught by the whoever called this function
```

Then you can use this like so:

```
db, c = connect()
```

```
13     # Open a cursor to perform database operations
14     cursor = conn.cursor()
15     # Query the database and obtain data as Python objects
16     cursor.execute("""
17         select articles.title, count(*) as views
18         from articles left join log
19         on concat('/article/',articles.slug)=log.path
20         where path like '/article/%'
21         group by log.path, articles.title order by views desc limit 3;
22         """)
23     popular_articles = cursor.fetchall()
24     # Close communication with the database
25     cursor.close()
26     conn.close()
27     return popular_articles
28
29
```

```
def get_most_popular_authors():
    """gets the authors with the most page views,
    when you sum up all of the articles each author has written.
    givess a sorted list with the most popular author at the top."""
    # Connect to an existing database
    conn = psycopg2.connect("dbname=%s user=%s" % (DB_NAME, DB_USER))
    # Open a cursor to perform database operations
    cursor = conn.cursor()
    # Query the database and obtain data as Python objects
    cursor.execute("""
        select authors.name, authors_score.views
        from authors left join authors_score
        on authors.id=authors_score.author_id order by views desc;
        """)
    popular_authors = cursor.fetchall()
    # Close communication with the database
    cursor.close()
    conn.close()
    return popular_authors
```

SUGGESTION

Notice how each function is very similar. You can refactor your code into a function that does the following:

```
def get_query_results(query)
    # connect to database
    # grab cursor
    # execute
    # commit
    # close
    # return results
```

Then you call it like so:

```
results = get_query_results("Send in query")
# print the results based on which query you're doing.
```

in each of your functions.

```

def get_down_times():
    """gets the days when more than 1% of requests lead to errors"""
    # Connect to an existing database
    conn = psycopg2.connect("dbname=%s user=%s" % (DB_NAME, DB_USER))
    # Open a cursor to perform database operations
    cursor = conn.cursor()
    # Query the database and obtain data as Python objects
    cursor.execute("select * from error_rates where error_rate>1.0")
    down_times = cursor.fetchall()
    # Close communication with the database
    cursor.close()
    conn.close()
    return down_times


def write_article_info(articles):
    """writes article-related information in report-file.txt"""
    report_file = open("report-file.txt", "a")
    report_file.write("MOST POPULAR ARTICLES:\n")

    for article in articles:
        report_file.write("\"")
        report_file.write(article[0])
        report_file.write("\" : ")
        report_file.write(str(article[1]))
        report_file.write(" views\n")

    report_file.write("\n\n\n")
    report_file.close()


def write_author_info(authors):
    """writes author-related information in report-file.txt"""
    report_file = open("report-file.txt", "a")
    report_file.write("MOST POPULAR AUTHORS:\n")

    for author in authors:
        report_file.write(author[0])
        report_file.write(" : ")
        report_file.write(str(author[1]))
        report_file.write(" views\n")

    report_file.write("\n\n\n")
    report_file.close()


def write_down_time_info(down_times):
    """writes error-rate information in report-file.txt"""
    report_file = open("report-file.txt", "a")
    report_file.write("DOWN TIME INFO:\n")

    for down_time in down_times:
        report_file.write(down_time[0])
        report_file.write(" : ")
```

```
105         report_file.write(str(down_time[1]))
106         report_file.write("% errors\n")
107
108     report_file.write("\n\n\n")
109     report_file.close()
110
111 # check __name__ variable so that code only executes,
112 # when you want to run the module as a program
113 # and NOT have it execute when someone just wants to import your module
114 if __name__ == "__main__":
115     write_article_info(get_most_popular_articles())
116     write_author_info(get_most_popular_authors())
117     write_down_time_info(get_down_times())
```

AWESOME

Looks great!

```
118
```

▸ report-file.txt

▸ README.md

RETURN TO PATH

Rate this review