Solutions - Chapter 17

17-1: Other Languages

Modify the API call in *python\_repos.py* so it generates a chart showing the most popular projects in other languages. Try languages such as *JavaScript*, *Ruby*, *C*, *Java*, *Perl*, *Haskell*, and *Go*.

***Note:****The code for this exercise includes an update that keeps a project with an empty description from breaking the program, and moves the font size settings to my\_style.*

import requests

import pygal

from pygal.style import LightColorizedStyle **as** LCS, LightenStyle **as** LS

*# Make an API call, and store the response.*

url **=** 'https://api.github.com/search/repositories?q=language:javascript&sort=stars'

r **=** requests**.**get(url)

**print**("Status code:", r**.**status\_code)

*# Store API response in a variable.*

response\_dict **=** r**.**json()

**print**("Total repositories:", response\_dict['total\_count'])

*# Explore information about the repositories.*

repo\_dicts **=** response\_dict['items']

names, plot\_dicts **=** [], []

**for** repo\_dict **in** repo\_dicts:

names**.**append(repo\_dict['name'])

*# When a project is removed, it's still listed with stars.*

*# So it's in the top projects, but has no description. The description*

*# is None, which causes an exception when being used as a label.*

**if** repo\_dict['description']:

desc **=** repo\_dict['description']

**else**:

desc **=** 'No description provided.'

plot\_dict **=** {

'value': repo\_dict['stargazers\_count'],

'label': desc,

'xlink': repo\_dict['html\_url'],

}

plot\_dicts**.**append(plot\_dict)

*# Make visualization.*

my\_style **=** LS('#333366', base\_style**=**LCS)

my\_style**.**title\_font\_size **=** 24

my\_style**.**label\_font\_size **=** 14

my\_style**.**major\_label\_font\_size **=** 18

my\_config **=** pygal**.**Config()

my\_config**.**x\_label\_rotation **=** 45

my\_config**.**show\_legend **=** False

my\_config**.**truncate\_label **=** 15

my\_config**.**show\_y\_guides **=** False

my\_config**.**width **=** 1000

chart **=** pygal**.**Bar(my\_config, style**=**my\_style)

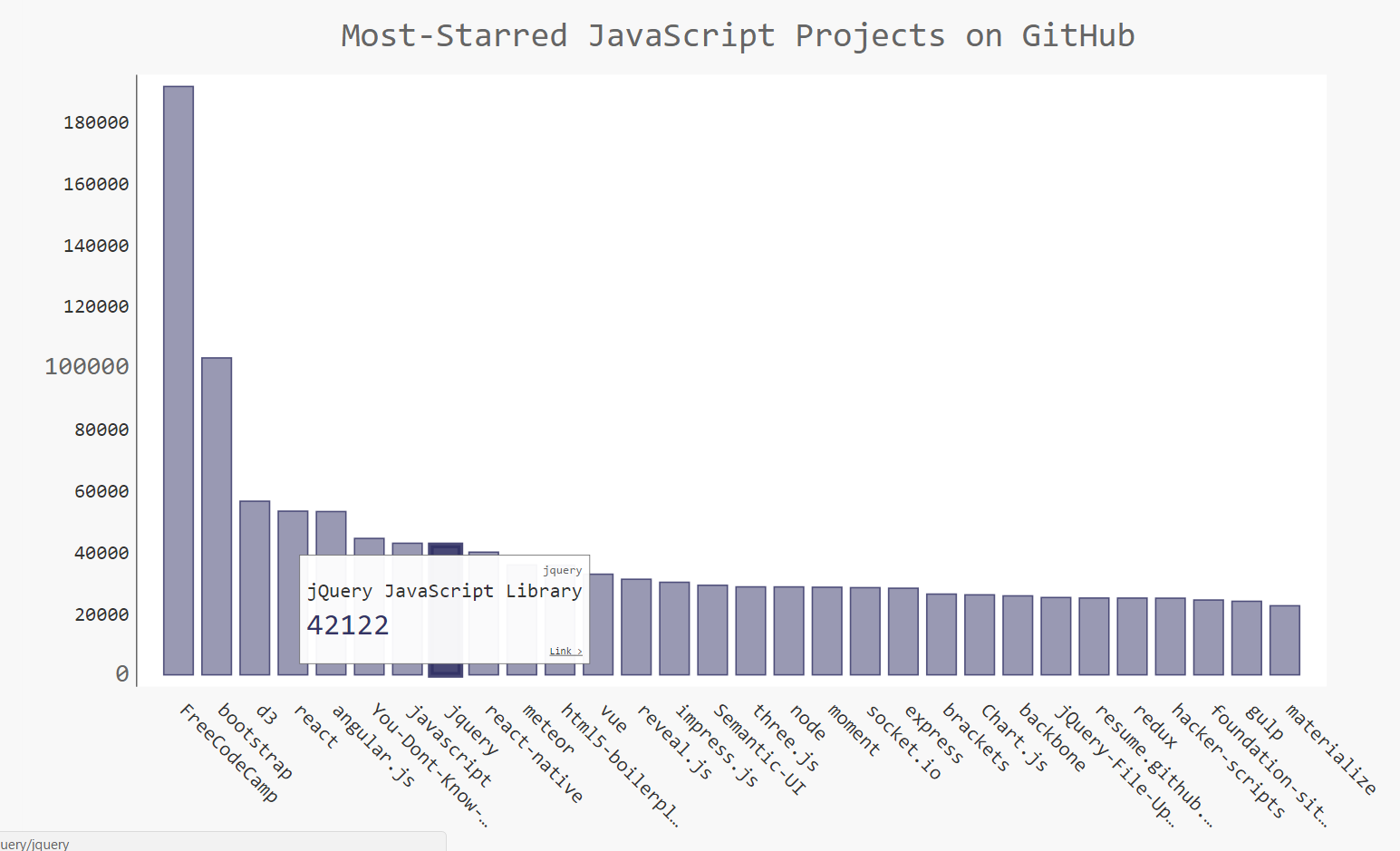
chart**.**title **=** 'Most-Starred JavaScript Projects on GitHub'

chart**.**x\_labels **=** names

chart**.**add('', plot\_dicts)

chart**.**render\_to\_file('js\_repos.svg')

Output:



17-2: Active Discussions

Using the data from *hn\_submissions.py*, make a bar chart showing the most active discussions currently happening on Hacker News. The height of each bar should correspond to the number of comments each submission has. The label for each bar should include the submission’s title, and each bar should act as a link to the discussion page for that submission.

import requests

import pygal

from pygal.style import LightColorizedStyle **as** LCS, LightenStyle **as** LS

from operator import itemgetter

*# Make an API call, and store the response.*

url **=** 'https://hacker-news.firebaseio.com/v0/topstories.json'

r **=** requests**.**get(url)

**print**("Status code:", r**.**status\_code)

*# Process information about each submission.*

submission\_ids **=** r**.**json()

submission\_dicts **=** []

**for** submission\_id **in** submission\_ids[:30]:

*# Make a separate API call for each submission.*

url **=** ('https://hacker-news.firebaseio.com/v0/item/' **+**

str(submission\_id) **+** '.json')

submission\_r **=** requests**.**get(url)

**print**(submission\_r**.**status\_code)

response\_dict **=** submission\_r**.**json()

submission\_dict **=** {

'title': response\_dict['title'],

'link': 'http://news.ycombinator.com/item?id=' **+** str(submission\_id),

'comments': response\_dict**.**get('descendants', 0)

}

submission\_dicts**.**append(submission\_dict)

submission\_dicts **=** sorted(submission\_dicts, key**=**itemgetter('comments'),

reverse**=**True)

**for** submission\_dict **in** submission\_dicts:

**print**("\nTitle:", submission\_dict['title'])

**print**("Discussion link:", submission\_dict['link'])

**print**("Comments:", submission\_dict['comments'])

titles, plot\_dicts **=** [], []

**for** submission\_dict **in** submission\_dicts:

titles**.**append(submission\_dict['title'])

plot\_dict **=** {

'value': submission\_dict['comments'],

'label': submission\_dict['title'],

'xlink': submission\_dict['link'],

}

plot\_dicts**.**append(plot\_dict)

*# Make visualization.*

my\_style **=** LS('#333366', base\_style**=**LCS)

my\_style**.**title\_font\_size **=** 24

my\_style**.**label\_font\_size **=** 14

my\_style**.**major\_label\_font\_size **=** 18

my\_config **=** pygal**.**Config()

my\_config**.**x\_label\_rotation **=** 45

my\_config**.**show\_legend **=** False

my\_config**.**truncate\_label **=** 15

my\_config**.**show\_y\_guides **=** False

my\_config**.**width **=** 1000

my\_config**.**y\_title **=** 'Number of Comments'

chart **=** pygal**.**Bar(my\_config, style**=**my\_style)

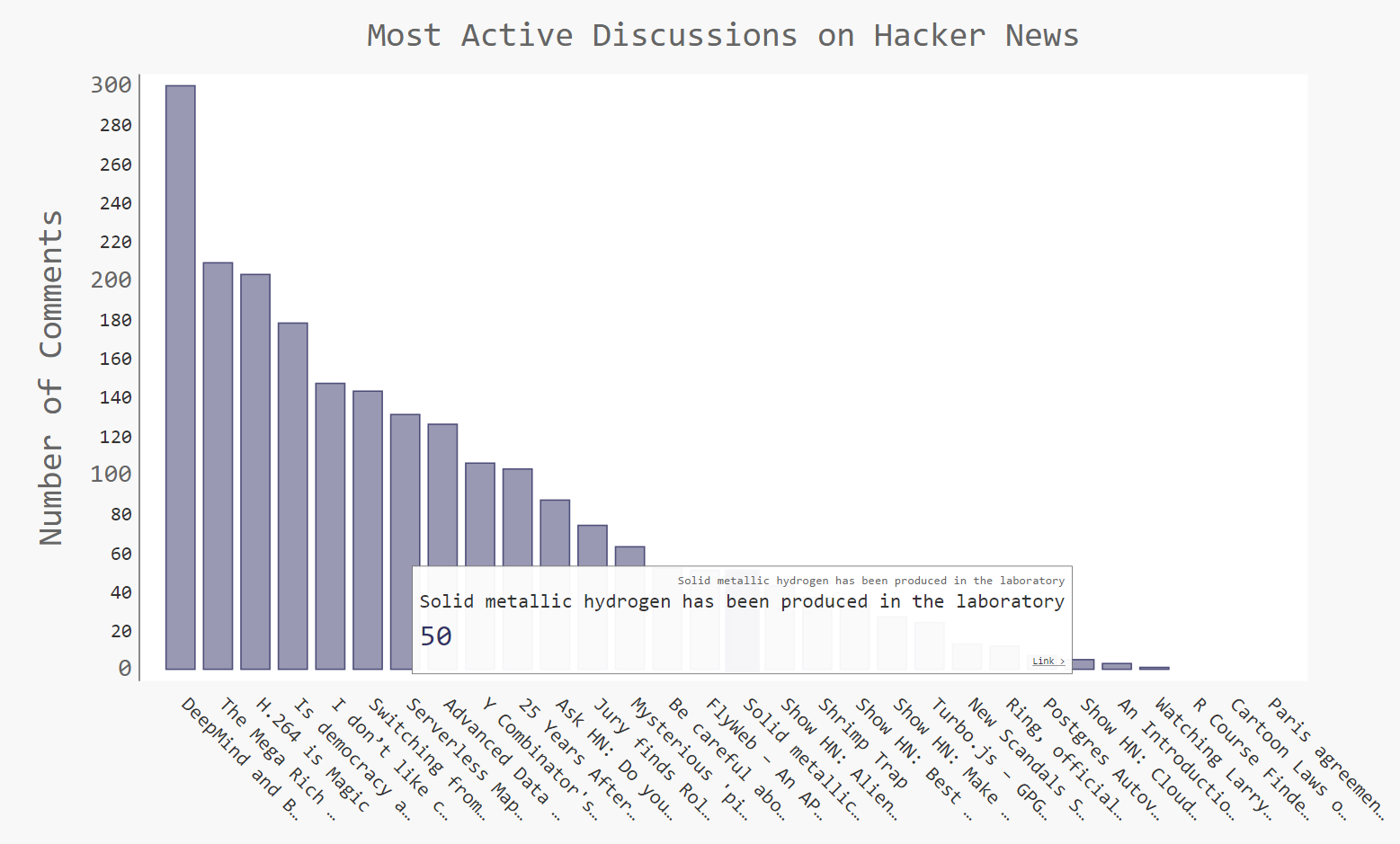
chart**.**title **=** 'Most Active Discussions on Hacker News'

chart**.**x\_labels **=** titles

chart**.**add('', plot\_dicts)

chart**.**render\_to\_file('hn\_discussions.svg')

Output:



17-3: Testing *python\_repos.py*

In *python\_repos.py*, we printed the value of status\_code to make sure the API call was successful. Write a program called *test\_python\_repos.py*, which uses unittest to assert that the value ofstatus\_code is 200. Figure out some other assertions you can make - for example, that the number of items returned is expected and that the total number of repositories is greater than a certain amount.

***Note:****The code for this exercise includes an update that keeps a project with an empty description from breaking the program, and moves the font size settings to my\_style.*

Writing tests pushes you to structure your code in a way that it can be tested. Here’s a revised version of *python\_repos.py*, with all of the work written as four functions:

import requests

import pygal

from pygal.style import LightColorizedStyle **as** LCS, LightenStyle **as** LS

**def** **get\_response**():

"""Make an api call, and return the response."""

url **=** 'https://api.github.com/search/repositories?q=language:python&sort=stars'

r **=** requests**.**get(url)

**return** r

**def** **get\_repo\_dicts**(response):

"""Return a set of dicts representing the most popular repositories."""

response\_dict **=** r**.**json()

repo\_dicts **=** response\_dict['items']

**return** repo\_dicts

**def** **get\_names\_plot\_dicts**(repo\_dicts):

"""Process the set of repository dicts, and pull out data for plotting."""

names, plot\_dicts **=** [], []

**for** repo\_dict **in** repo\_dicts:

names**.**append(repo\_dict['name'])

*# Some projects lack a description, which causes an error when*

*# labeling bars. Specify a label if there's no description.*

description **=** repo\_dict['description']

**if** **not** description:

description **=** "No description provided."

plot\_dict **=** {

'value': repo\_dict['stargazers\_count'],

'label': description,

'xlink': repo\_dict['html\_url'],

}

plot\_dicts**.**append(plot\_dict)

**return** names, plot\_dicts

**def** **make\_visualization**(names, plot\_dicts):

"""Make visualization of most popular repositories."""

my\_style **=** LS('#333366', base\_style**=**LCS)

my\_style**.**title\_font\_size **=** 24

my\_style**.**label\_font\_size **=** 14

my\_style**.**major\_label\_font\_size **=** 18

my\_config **=** pygal**.**Config()

my\_config**.**x\_label\_rotation **=** 45

my\_config**.**show\_legend **=** False

my\_config**.**truncate\_label **=** 15

my\_config**.**show\_y\_guides **=** False

my\_config**.**width **=** 1000

chart **=** pygal**.**Bar(my\_config, style**=**my\_style)

chart**.**title **=** 'Most-Starred Python Projects on GitHub'

chart**.**x\_labels **=** names

chart**.**add('', plot\_dicts)

chart**.**render\_to\_file('python\_repos.svg')

r **=** get\_response()

repo\_dicts **=** get\_repo\_dicts(r)

names, plot\_dicts **=** get\_names\_plot\_dicts(repo\_dicts)

make\_visualization(names, plot\_dicts)

Now we can write tests for these functions. Here we test that we get a response with a status code of 200, and we test that some of the keys we expect to find in each repository’s dictionary are in the first project’s dictionary.

import unittest

import python\_repos\_for\_testing **as** pr

**class** **PythonReposTestCase**(unittest**.**TestCase):

"""Tests for python\_repos.py."""

**def** **setUp**(self):

"""Call all the functions here, and test elements separately."""

self**.**r **=** pr**.**get\_response()

self**.**repo\_dicts **=** pr**.**get\_repo\_dicts(self**.**r)

self**.**repo\_dict **=** self**.**repo\_dicts[0]

self**.**names, self**.**plot\_dicts **=** pr**.**get\_names\_plot\_dicts(self**.**repo\_dicts)

**def** **test\_get\_response**(self):

"""Test that we get a valid response."""

self**.**assertEqual(self**.**r**.**status\_code, 200)

**def** **test\_repo\_dicts**(self):

"""Test that we're getting the data we think we are."""

*# We should get dicts for 30 repositories.*

self**.**assertEqual(len(self**.**repo\_dicts), 30)

*# Repositories should have required keys.*

required\_keys **=** ['name', 'owner', 'stargazers\_count', 'html\_url']

**for** key **in** required\_keys:

self**.**assertTrue(key **in** self**.**repo\_dict**.**keys())

unittest**.**main()

Output:

..

----------------------------------------------------------------------

Ran 2 tests in 1.969s

OK