ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ

SCHOOL OF BUSINESS

ΤΜΗΜΑ ΔΙΟΙΚΗΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ & ΤΕΧΝΟΛΟΓΙΑΣ

DEPARTMENT OF MANAGEMENT SCIENCE & TECHNOLOGY

Software Engineering in Practice

Final Report

Implementation Team

Python3s

Konstantinos Lolos 8160064

Ioannis Kokkosis 8160190

Project Understanding

We chose to contribute to an open source project called Jarvis. Jarvis is a personal assistant that runs in the most popular operating systems such as MacOS, Linux and Windows. The code base of Jarvis is implemented in python 2 and 3 programming language and its functionality and capabilities are based on various plugins that help the user do some exciting stuff. Jarvis can tell you the weather, he can find places, such as restaurants, near you, he can tell you a joke to cheer you up and so many other great things. Jarvis also comes up with some funny mini games such as akinator, blackjack etc. These features of Jarvis vary from basic utilities like weather forecast to more advanced like data processing for machine learning purposes. Finally, Jarvis has not any kind of user interface to interact with the user. The interaction between the User and Jarvis is through the terminal (for MacOS and Linux users) or the command line (for Windows users).

One great thing about Jarvis is that its capabilities are limitless due to its structure. Jarvis structure is very simple. The basic interaction in the command line, like calling Jarvis, exiting Jarvis, or calling a specific plugin, is based on some classes that are implemented in certain files like Jarvis.py, CmdInterpreter.py and PluginManager.py. The implementation and functionality of all the plugins is separated in a different folder called plugins. This folder contains all the files that are related to the functionality of the plugins and so this folder is responsible for the extra capabilities of Jarvis.

Building Jarvis is also a very simple process. If someone wants to install Jarvis to his/her local machine all he/she needs to do is to follow the detailed steps in README.md file that is placed in the root of the repository. The only prerequisites that someone needs to have is to have already install a version of python (version 2 or 3, works on both versions) and a version of python virtualenv (virtual environment).

Our Contribution to Jarvis

Our plan was to contribute to the source code of Jarvis, by finding a bug, and to implement a new plugin for Jarvis. While using Jarvis and playing with the plugins we noticed that a certain plugin was poorly implemented. That plugin was bulkresizer. The purpose of bulkresizer is to resize images into a given size with padding and it is specifically designed for Deep Learning and data collection process. The source code of bulkresizer was not implemented right. Bulkresizer was throwing too many unhandled exceptions, the welcome messages were printed out in the wrong order and many other problems that were obvious. As for the new feature an idea came to us as we studied the instructions and guidelines to start experimenting with plugins. The thought was to apply these guidelines into a template to help programmers starting to contribute to Jarvis. We also had some discoveries along the way.

Implementation Quality

Issue 1: Create plugin to create new_plugin.py with template

```python
"""The os.path method is used to track the path in which this plugin is stored
and locate the Jarvis/custom folder through relative pathing.
"""
PLUGINS_PATH = os.path.dirname(os.path.abspath(__file__))
CUSTOM_PLUGINS_PATH = os.path.join(PLUGINS_PATH, '..', '..', 'custom/')


@require(platform=MACOS)
@plugin("create plugin")
def create_plugin_MAC(jarvis, s):

    # Jarvis asks for the name of the plugin to create if not provided.
    if s == "":
        jarvis.say("Please insert the name of your plugin: ", Fore.RED)
        name = jarvis.input()
    else:
        name = s
    """The name fillters through the format_file name
    which converts it to a valid filename.
    """
    filename = format_filename(name)
    # Flag to terminate process
    exit = False
    """Checks if file already exists in either the plugins main folder
    or in the custom plugins folder and also asks the user if he wants to exit
    """
    while(os.path.isfile(CUSTOM_PLUGINS_PATH + filename + ".py") or
          os.path.isfile(PLUGINS_PATH + "/" + filename + ".py") and
          not exit):
        jarvis.say("A plugin with the name '" + filename +
                   "' already exists", Fore.RED)
        jarvis.say("Please choose another name or type 'exit' " +
                   "for obvious results:", Fore.CYAN)

        new_name = jarvis.input()
        if new_name == 'exit':
            exit = True
        filename = format_filename(new_name)
    if(not exit):
        """The templated is generated through the create_template funcion
        and is exceuted through the os.system method.
        """
        string = create_template(CUSTOM_PLUGINS_PATH, filename)
        os.system(string)

        if file_exists(filename):
            jarvis.say(filename + ".py created successfully inside " +
                       "Jarvis/custom", Fore.CYAN)
            string = "open " + CUSTOM_PLUGINS_PATH + filename + ".py"
            os.system(string)
        else:
            jarvis.say("Something went wrong in the creation of the plugin :(",
                       Fore.RED)
```

- That is the form of the plugin itself that controls the interaction of the user and call the appropriate functions.
- Pep8 guidelines are strictly followed as a principle and checked (Jarvis's Travis is set with a flake8 command)
- The @require tag is used because some os commands differ between macOS and Linux**.**
- Documentation is provided regarding all the utilities for any future maintenance needed.

```python
def file_exists(filename):
    """This method is used to check if a file with
    the name "filename" exists in the Jarvis/custom folder
    """
    if os.path.isfile(CUSTOM_PLUGINS_PATH + filename + ".py"):
        return True
    else:
        return False
```

- There is also a Linux version but for the sake of space only this one is displayed here. (the only difference is the command used to open the file)
- The **file_exists** method utilizes the isfile function to check if a file exists and returns a Boolean value.

```python
def create_template(path, filename):
    """This method is used to format the template of the plugin
that is being created given the filename and the path
in which it will be stored.
"""
    template = """cd """ + path + """
            cat >> """ + filename + """.py << EOL
# All plugins should inherite from this library
from plugin import plugin

# This is the standard form of a plugin for jarvis

# Anytime you make a change REMEMBER TO RESTART Jarvis

# You can run it through jarvis by the name
# in the @plugin tag.


@plugin("my plugin")
def my_plugin(jarvis, s):

    # Prints 'This is my new plugin!'
    jarvis.say("This is my new plugin!")

# For more info about plugin development visit:
# https://github.com/sukeesh/Jarvis/blob/master/doc/PLUGINS.md
EOL"""

    return template
```

- The **create_template** function returns a string with the template of the plugin that is being created

```python
def format_filename(name):

    """Take a string and return a valid filename constructed from the string.
Uses a whitelist approach: any characters not present in valid_chars are
removed. Also spaces are replaced with underscores.

Note: this method may produce invalid filenames such as ``, `.` or `..`
When I use this method I prepend a date string like '2009_01_15_19_46_32_'
and append a file extension like '.txt', so I avoid the potential of using
an invalid filename.
"""
    import string

    valid_chars = "-_.() %s%s" % (string.ascii_letters, string.digits)
    filename = ''.join(c for c in name if c in valid_chars)
    filename = filename.replace(' ', '_')  # I don't like spaces in filenames.
    return filename
```

- The **format_filename** function is kind of self-explanatory. It simply takes a string as an argument and clears the characters that are not “valid” such as exclamation marks(!), at characters (@) and replaces spaces with underscores. Its basic functionality is to give a proper name to the file that the user wants to create.

Issue 2: Make bulkresizer plugin more user friendly

```python
def bulk_resizer(input_path, output_path, desired_size=32,
                 color=[0, 0, 0], rename=True):
    filepath = list_contents(input_path)
    '''Resizes the images into a given size

    Creates a resized image for an existing image

    Parameters
    ----------

    input_path: an input_path (str)
        a path that leads to an existing dir that contains images
    output_path: an output_path (str)
        a path that will contain the resized images
    desired_size: a desired_size (int)
        a number that it is the size of the resized images
    color: list
        a list that represents the color of the border of the resized images.
        Color equal to [0, 0, 0] means that the border color will be black
    rename: bool
        if this variable is set to True then the resized images names will be
        set to an non repeating whole number series. If it is set to False the
        resized images will have the same name with the original ones.
    '''

    for im_pth in filepath:

        im = cv2.imread(im_pth)
        old_size = im.shape[:2]
        ratio = float(desired_size) / max(old_size)
        new_size = tuple([int(x * ratio) for x in old_size])

        # new_size should be in (width, height) format

        im = cv2.resize(im, (new_size[1], new_size[0]))

        delta_w = desired_size - new_size[1]
        delta_h = desired_size - new_size[0]
        top, bottom = delta_h // 2, delta_h - (delta_h // 2)
        left, right = delta_w // 2, delta_w - (delta_w // 2)

        new_im = cv2.copyMakeBorder(im, top, bottom, left, right,
                                    cv2.BORDER_CONSTANT, value=color)
        if rename:
            output_path1 = rename_img(output_path, filepath.index(im_pth))
        else:
            output_path1 = output_path_concat(output_path, im_pth)
        cv2.imwrite(output_path1, new_im)
```

- The bulk_resize function resizes images into a given size.
- This function uses cv2 module to process the images.
- It reads the images file paths that are stored in a list and then it resizes them into a new size. Finally, it creates a border around the images to look like a photo frame.
- With the use of two other functions it concatenates the output file paths of the resized images
- Finally, it writes the resized images into the output path, using the imwrite functions of cv2 module

```python
def valid_path(path):
    '''Checks if a given path leads to a valid directory

    Returns true if the path leads to valid dir, false otherwise

    Parameters
    ----------

    path: a path (str)
        a string variable that represents a path
    '''
    return True if os.path.isdir(path) else False
```

- The valid_path function just checks if a path leads to a valid directory using the isdir function of os module. It returns true if the path leads to a valid directory, false otherwise.

```python
def dir_exist(path):
    '''Checks if a directory path exists

    Returns true if the dir path exists, false otherwise

    Parameters        You, a month ago • Update bulkresize.
    ----------

    path: a path (str)
        a string that represents a path
    '''
    return True if os.path.exists(path) else False
```

- The dir_exist function checks if a given path exists using the exists function of os module. It returns true if the path exists, false otherwise

```python
def create_dir(path):
    '''Creates a new directory

    Returns nothing. This function simply creates a new dir

    Parameters
    ----------

    path: a path (str)
        a string that represents the path of the dir that will be created
    '''
    os.makedirs(path)
```

- The create_dir function creates a new directory in the computer, using the makedirs function of os module

```python
def list_contents(input_path):
    '''Lists all the image files of a given path direxory

    Returns a list that contains only the image files of given path directory

    Parameters
    ----------

    input_path: a path (str)
        a string that represents a path that leads to a valid dir
    '''
    filepath = list()
    filename = os.listdir(input_path)

    for name in filename:
        path = input_path + "/" + name
        if os.path.isfile(path) and get_extension(path):
            filepath.append(path)
    return filepath
```

- The list_contents function, stores the contents of directory in a list. The get_extension function is used to separate image files from other files. In other words, list_contents stores image file paths in a list.

```python
def remove_backslash(path):
    '''Removes all the backslashes from a path and replace them with spaces

    Returns a new path without backslashes

    Parameters
    ----------

    path: a path (str)
        a string that represents a path that leads to a valid dir
    '''
    if '\\ ' in path:
        path = path.replace('\\ ', ' ')
    return path
```

- The remove_backslash function removes all the backslashes from a string and replaces them with whitespaces.

```python
def get_extension(path):
    '''Checks if an extension of a file path is an image using IMAGE_FORMATS list

    Returns true if the extension of the file path represents
    an image, false otherwise

    Parameters
    ----------

    path: a path (str)
        a string that leads to a file path
    '''
    file_extension = os.path.splitext(path)[1]

    if file_extension in IMAGE_FORMATS:
        return True
    else:
        return False
```

- The get_extension function checks if a file path represents an image. If it is returns true, otherwise returns false.

```python
def rename_img(path, number):
    '''Renames a file path of an image

    Returns a new name of the file path for the resized images

    Parameters        You, a month ago • Update bulkresize.py file
    ----------

    path: a path (str)
        a string that leads to an existing file path
    number: a number (int)
        a number that is used in the concatination for the image rename
    '''
    output_path = path + '/' + str(number) + '.jpg'
    return output_path
```

- The rename_img function renames the resized images to non-repeating whole number series.

```python
def output_path_concat(path, im_path):
    '''Creates a file path of an image

    Returns an output file path for the resized images

    Parameters
    ----------
    path: a path (str)
        a string that lead to an existing file path
    im_path: an image path (str)
        a string that leads to an existing image file path
    '''
    output_path = path + '/' + \
        os.path.splitext(os.path.basename(im_path))[0] + '.jpg'
    return output_path
```

- The output_path_concat function concatenates the output file path of a resized image.

```python
@plugin("bulkresizer")
def spin(jarvis, s):
    """
    This resizes all the images in a given directory
    to given size and renames them, Specially designed for Deep Leanring
    data collection process.

    """
    answer = ""
    jarvis.say(' ')
    jarvis.say(
        'This is bulk resizer. ' +
        'Bulkresizer is a plugin that resizes images' +
        'into a given size with padding!!!', Fore.BLUE)
    jarvis.say(
        'Specially designed for Deep Learning ' +
        'and data collection process.', Fore.BLUE)
    jarvis.say(' ')
    jarvis.say(
        'Enter the path of directory with images to be resized : ',
        Fore.BLUE)
    path1 = jarvis.input()
    path1 = remove_backslash(path1)
    while not valid_path(path1):
        jarvis.say(
            'The path ' + path1 +
            ' does not lead to a directory!', Fore.RED)
        jarvis.say(
            'Please enter a path that leads to an EXISTING DIRECTORY.',
            Fore.RED)
        path1 = jarvis.input()
    jarvis.say(
        'Should I rename them to non repeating whole number series?',
        Fore.YELLOW)
    jarvis.say('Press y for "YES" n for "NO"', Fore.YELLOW)
    rename = jarvis.input()
    jarvis.say('Enter the path of output directory :', Fore.YELLOW)
    path2 = jarvis.input()
    if not dir_exist(path2):
        jarvis.say(
            'The path ' + path2 + ' does not exist. Do you want to create it?',
            Fore.YELLOW)
        jarvis.say('Print y for "YES" n for "NO"', Fore.YELLOW)
        answer = jarvis.input()
    if answer is 'y':
        create_dir(path2)
    else:
        while not dir_exist(path2):
            jarvis.say(
                'The output path does not exist. ' +
                'Please type an existing path!',
                Fore.YELLOW)
            path2 = jarvis.input()
    jarvis.say("Enter the target size :", Fore.YELLOW)
    size = jarvis.input_number(rtype=int)
    if rename == 'y':
        bulk_resizer(path1, path2, size, [0, 0, 0], True)
    else:
        bulk_resizer(path1, path2, size, [0, 0, 0], False)
    jarvis.say("Resizing Compleated!! Thank you for using jarvis", Fore.GREEN)
```

- The spin function is responsible for the execution of bulkresizer functionality
- First it asks the user to give a path that leads to a directory with images
- If the user types an invalid path, bulkresizer informs him/her by telling him/her to give a path that leads to a valid directory.
- Then bulkresizer asks the user if he/she wants to rename the resized images to a non-repeating whole number series.
- Then asks the user to give an output path. If the output path does not exist, bulkresizer ask the user if he/she wants to create the output path
- Finally, bulkresizer asks the user the target size with which it will resize the images.

Before vs After our implementation to bulkresizer

In the following images you can see the old version of the source code of bulkresizer plugin. The purpose of these images is to compare the differences between the old and the new version above. In the old version all the functionality of the plugin was implemented in only two functions. The bulk_resize functions was executing several functionalities which made the implementation of testing code an extremely hard and painful process. Now with the new version, every single function executes a certain functionality, which makes the code more readable, understandable, clear, and testable.

```python
def bulk_resizer(input_path, output_path, desired_size=32,
                 color=[0, 0, 0], rename=True):
    img_no = 0

    filename = os.listdir(input_path)
    filepath = []
    for name in filename:
        path = input_path + "/" + name
        if os.path.isfile(path):
            filepath.append(path)
    for im_pth in filepath:
        try:
            im = cv2.imread(im_pth)
            old_size = im.shape[:2]

        except:
            continue
        ratio = float(desired_size) / max(old_size)
        new_size = tuple([int(x * ratio) for x in old_size])

        # new_size should be in (width, height) format

        im = cv2.resize(im, (new_size[1], new_size[0]))

        delta_w = desired_size - new_size[1]
        delta_h = desired_size - new_size[0]
        top, bottom = delta_h // 2, delta_h - (delta_h // 2)
        left, right = delta_w // 2, delta_w - (delta_w // 2)

        new_im = cv2.copyMakeBorder(im, top, bottom, left, right,
                                    cv2.BORDER_CONSTANT, value=color)
        if rename:
            output_path1 = output_path + "/" + str(img_no) + ".jpg"
            img_no += 1
        else:
            output_path1 = output_path + "/" + \
                os.path.splitext(os.path.basename(im_pth))[0] + ".jpg"
        cv2.imwrite(output_path1, new_im)
```

```python
@plugin("bulkresizer")
def spin(jarvis, s):
    """
    \nThis resizes all the images in a given directory
    to given size and renames them, Specially designed for Deep Leanring
    data collection process.
    """
    jarvis.say(' ')
    jarvis.say('This is bulk resizer it will resize the image into given',
               'size with padding !!!! Specially designed for Deep Leanring',
               ' data collection process')
    jarvis.say('Enter the path of directory with images to be resized : ')
    path1 = jarvis.input()
    jarvis.say('Should I rename them to non repeating whole number series?',
               'y/n :')
    rename = jarvis.input()
    jarvis.say('Enter the path of output directory :')
    path2 = jarvis.input()
    jarvis.say("Enter the target size :")
    size = jarvis.input_number(rtype=int)
    if rename == 'y':
        bulk_resizer(path1, path2, size, [0, 0, 0], True)
    else:
        bulk_resizer(path1, path2, size, [0, 0, 0], False)
    jarvis.say("Resizing Compleated!! Thank you for using jarvis")
```

Issue 3: queue_input method works more like a stack rather than a queue.

This issue was something we discovered while running some tests for bulkresizer. The bug was in the construction of a variable called _input_queue. The role of this variable is to store input to feed into Jarvis during an integration test that needed multiple inputs from the user. From the name of it we expect it to act like a queue and so did the collaborator who wrote the code. It turned out that it did not. The variable was acting like a stack. That meant that you needed to add items in the list in the opposite order you needed them to be fed into Jarvis. An unnoticed bug because according to the main collaborator there was not a test until now in which order mattered.

The implementation of that issue was extremely easy. We just imported the deque module from collections library, we initialized the input_queue from a simple list to a deque and we changed the extraction method of the elements of the queue from pop() to popleft().

```
import unittest
from functools import partial
from CmdInterpreter import JarvisAPI
from collections import deque
```

```
self.data = {}
self._input_queue = []
self._input_queue = deque()
self.is_voice_enabled = False
```

```
def input(self, prompt='', color=''):
    if len(self._input_queue) == 0:
        raise BaseException("MockJarvisAPI: No predefined answer in queue - add answer with 'self.queue_input(\"TEXT\")'")
    return self._input_queue.pop()
    return self._input_queue.popleft()
```

Testing and Continuous Integration

We wrote testing code only for the issues of create_plugin and bulkresizer plugin. We used individual unit tests and integration tests to test that both plugins work properly and produce the results that we anticipated. In the following lines we analyze with further details the tests that we decided to implement.

Issue 1: Create plugin to create new_plugin.py with template

```python
import unittest
from unittest import mock
from tests import PluginTest
from plugins import create_plugin
from plugins.create_plugin import create_plugin_MAC
from plugins.create_plugin import create_plugin_LINUX


class create_pluginTest(PluginTest):

    def setUp(self):
        self.mac_module = self.load_plugin(create_plugin_MAC)
        self.linux_module = self.load_plugin(create_plugin_LINUX)

    @unittest.mock.patch('os.system')
    def test_create_file_MAC(self, os_system):
        self.mac_module.run("my_test")
        output = self.history_say().last_text()
        # Pull the template from the main plugin
        template = create_plugin.create_template(create_plugin.CUSTOM_PLUGINS_PATH, "my_test")
        # Mock is called excactly once because the file was never created so it will not open
        os_system.assert_called_once_with(template)

    @unittest.mock.patch('os.system')
    def test_create_file_LINUX(self, os_system):
        self.linux_module.run("my_test")
        output = self.history_say().last_text()
        # Pull the template from the main plugin
        template = create_plugin.create_template(create_plugin.CUSTOM_PLUGINS_PATH, "my_test")
        # Mock is called excactly once because the file was never created so it will not open
        os_system.assert_called_once_with(template)

    def test_format_filename(self):
        actual = create_plugin.format_filename("my test!@#")
        expected = "my_test"
        self.assertEqual(actual, expected)

    def test_file_exists_True(self):
        actual = create_plugin.file_exists("test")
        self.assertTrue(actual)

    def test_file_exists_False(self):
        actual = create_plugin.file_exists("ghost")
        self.assertFalse(actual)


if __name__ == '__main__':
    unittest.main()
```

- As far as testing is concerned **two integration tests** were implemented for each version of the plugin (macOS and Linux) and some unit tests for the different functions.
- For the integration tests **mocking** was utilized so the file was not being created/ opened every time the tests ran.

Isuue2: Make bulkresizer plugin more user friendly

```python
class Bulkresize(PluginTest):

    def setUp(self):
        self.bulkresize_module = self.load_plugin(spin)

    def test_valid_path(self):
        valid_test_dir = os.path.join(DATA_PATH, 'images')
        actual = bulkresize.valid_path(valid_test_dir)
        self.assertTrue(actual)

    def test_invalid_path(self):
        invalid_test_dir = os.path.join(DATA_PATH, 'test_invalid_dir')
        actual = bulkresize.valid_path(invalid_test_dir)
        self.assertFalse(actual)

    def test_dir_exist(self):
        dir_exist = os.path.join(DATA_PATH, 'images')
        actual = bulkresize.dir_exist(dir_exist)
        self.assertTrue(actual)

    def test_dir_not_exist(self):
        dir_not_exist = os.path.join(DATA_PATH, 'test_invalid_dir')
        actual = bulkresize.dir_exist(dir_not_exist)
        self.assertFalse(actual)
```

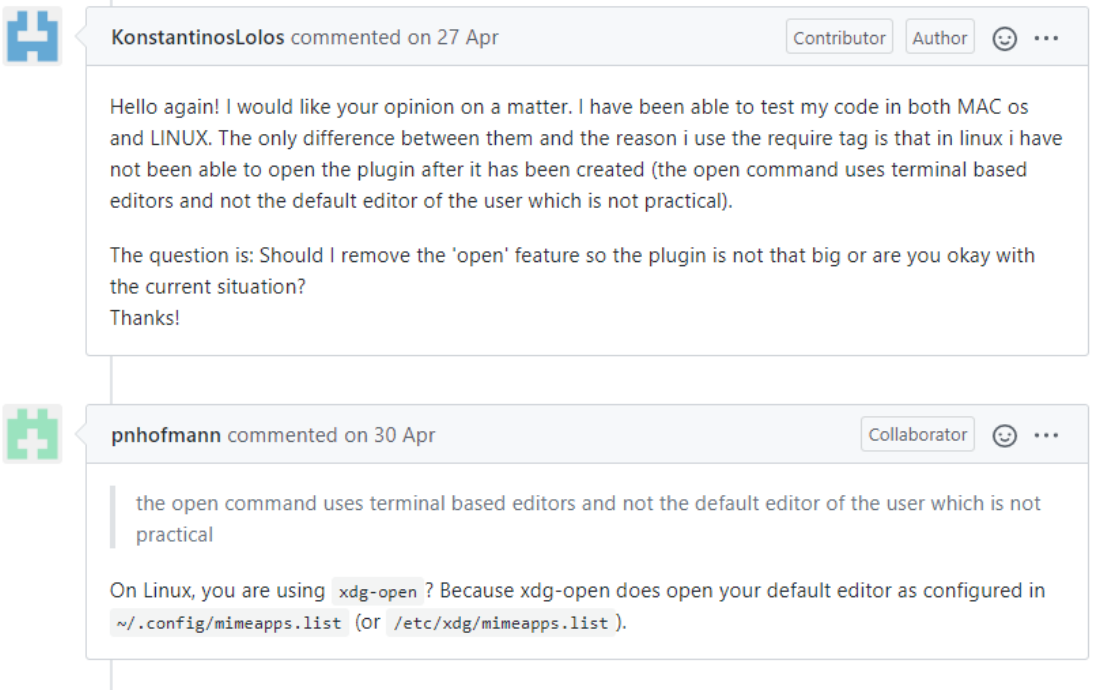
```python
def test_list_contents(self):
    expected_list = [DATA_PATH + 'images/' + 'dummy-man.jpg']
    test_path = os.path.join(DATA_PATH, 'images')
    actual = bulkresize.list_contents(test_path)
    self.assertListEqual(actual, expected_list)

def test_remove_backslash(self):
    path_str = '/jarvis/jarviscli/plugin\\ name'
    expected = '/jarvis/jarviscli/plugin name'
    actual = bulkresize.remove_backslash(path_str)
    self.assertEqual(actual, expected)

@unittest.mock.patch('os.makedirs')
def test_create_dir(self, os_makedirs):
    path = os.path.join(DATA_PATH, 'dir_to_be_created')
    bulkresize.create_dir(path)
    os_makedirs.assert_called_once_with(path)

def test_rename_img(self):
    num = 1
    test_path = 'jarvis/jarviscli/plugins'
    expected = 'jarvis/jarviscli/plugins/1.jpg'
    actual = bulkresize.rename_img(test_path, num)
    self.assertEqual(actual, expected)
```

```python
def test_output_path_concat(self):
    test_path = 'jarvis/jarviscli/plugins'
    test_image_name = 'image.jpg'
    expected = 'jarvis/jarviscli/plugins/image.jpg'
    actual = bulkresize.output_path_concat(test_path, test_image_name)
    self.assertEqual(actual, expected)

def test_get_extension_true(self):
    test_path = 'jarvis/jarviscli/plugins/image.jpg'
    actual = bulkresize.get_extension(test_path)
    self.assertTrue(actual)

def test_get_extension_false(self):
    test_path = 'jarvis/jarviscli/plugins/image.py'
    actual = bulkresize.get_extension(test_path)
    self.assertFalse(actual)

def test_spin(self):
    self.queue_input(DATA_PATH + 'images/')
    self.queue_input('y')
    self.queue_input(DATA_PATH + 'images/')
    self.queue_input('200')
    self.bulkresize_module.run(' ')
    actual = self.history_say().last_text()
    excepted = 'Resizing Compleated!! Thank you for using jarvis'
    self.assertEqual(actual, excepted)
    os.remove(DATA_PATH + 'images/0.jpg')
```

- The testing code for bulkresizer plugin was a simple task due to the implementation that we made for the bulkresizer plugin.
- The testing code was concerned from 11 small and individual unit tests that test the functionality of various methods.
- The testing code contains one integration test for the create_dir function, mocking was utilized so that the folder was not being created every time the tests ran.

Collaboration with the development team of Jarvis

Working and communicating with the development team was something that made us feel a little bit anxious because we thought that if we were facing a problem, the help would be quite limited. However, the collaboration with the development team was excellent and a very pleasant experience. While we were developing the new feature for Jarvis (a plugin that creates a new_plugin.py file with template) we had to deal with some problems both in the implementation and the testing code. To solve these problems, we decided to ask the team for help and requested their opinion on how to overcome these difficulties we were encountering. Without any delay the main collaborator of Jarvis, pnhofman, helped us by providing some feeds to check, that were very useful to our problems. The discussions and the collaboration with the development team are provided with screenshots below.



KonstantinosLolos commented on 27 Apr — Contributor — Author

Hello again! I would like your opinion on a matter. I have been able to test my code in both MAC os and LINUX. The only difference between them and the reason i use the require tag is that in linux i have not been able to open the plugin after it has been created (the open command uses terminal based editors and not the default editor of the user which is not practical).

The question is: Should I remove the 'open' feature so the plugin is not that big or are you okay with the current situation?
Thanks!

pnhofmann commented on 30 Apr — Collaborator

> the open command uses terminal based editors and not the default editor of the user which is not practical

On Linux, you are using `xdg-open`? Because xdg-open does open your default editor as configured in `~/.config/mimeapps.list` (or `/etc/xdg/mimeapps.list`).

KonstantinosLolos commented on 30 Apr — Contributor — Author

Looks like thats what it does! Thanks for the tip! I added that in and made some more practical changes. I shall proceed with a pull request for review.

KonstantinosLolos mentioned this issue on 30 Apr

Add create_plugin.py plugin #684 — Merged

pnhofmann added a commit that referenced this issue on 13 May

`Add new plugin 'create_plugin.py' to create template for new plugin #674` ... `bca1b8b`

This was referenced on 13 May

Fix create_plugin.py + some various files to match with PEP8 guidelines #689 — Merged

Add test cases for create_plugin.py and minor changes in create_plugin.py #694 — Merged

KonstantinosLolos commented 26 days ago — Contributor

In this pull request:

- I have added some test cases for the create_plugin.py
- created an extra function to make the code cleaner.
- Fixed minor "too many blank lines issue in optional.py

I want also to add test cases for the plugin itself but I am having some trouble with mocking (I am now getting into it) and I cant seem to test the plugin without creating a file (as the plugin's utility is). Any tips as to where to look for info?
Thanks!
#674

pnhofmann commented 25 days ago — Collaborator

> and I cant seem to test the plugin without creating a file

There is in fact a mocking-module for file-operations! Docu with example code:

https://docs.python.org/3/library/unittest.mock.html#mock-open

Generally speaking, it is also possible to create a temporary directory (for example using https://docs.python.org/3/library/tempfile.html) and running test-code in this there.
But in your case; the mocking-solution is probably just more "elegant" ;).

KonstantinosLolos commented 21 days ago — Author — Contributor

Decided to go with mocking the os.System method so the file is not being created every time the tests run. What do you think?

pnhofmann commented 21 days ago — Collaborator

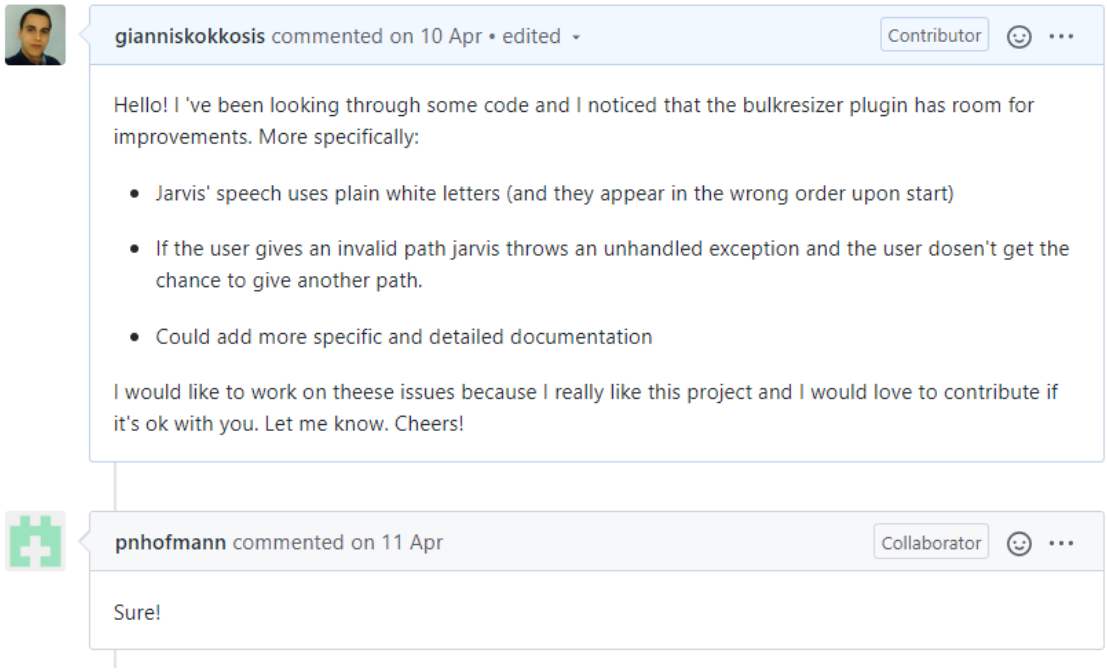
Nice ;)

pnhofmann merged commit `10c1d20` into `sukeesh:master` 21 days ago — Revert

Project Organization

For all the issues we worked with Git and GitHub on our forked repositories. We cloned our forked repositories to our local machines, we installed Jarvis and experimented with the functionalities that this project has, we looked up some code to see how this project is structured and implemented. Then we spotted some issues in certain pieces of the project and we opened some issues on GitHub. After, we implemented the fixes in our forked repositories by making small and understandable commits, to show what changes we made and why. Finally, we made several Pull Requests for our issues, informing the development team about what we are changing and what are the benefits of these changes. In the following images you can see all the issues, the Pull Requests, and some commit messages that we made.

Issue for Bulk Resizer Plugin:



gianniskokkosis commented on 10 Apr • edited — Contributor

Hello! I 've been looking through some code and I noticed that the bulkresizer plugin has room for improvements. More specifically:

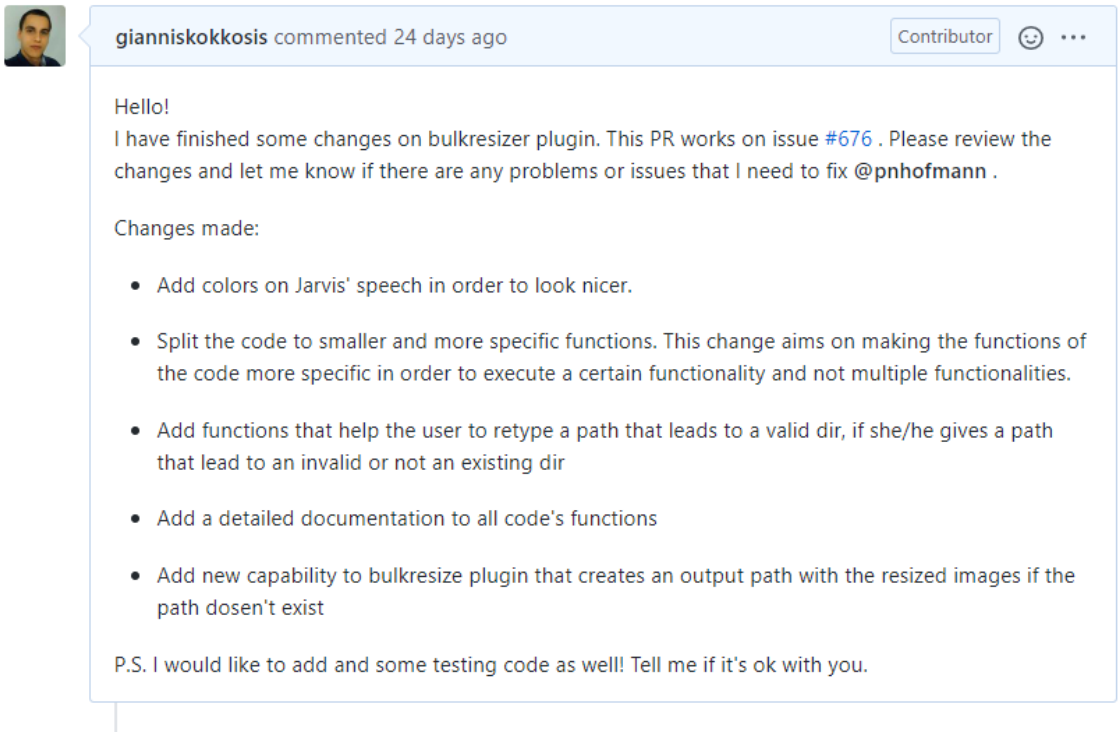
- Jarvis' speech uses plain white letters (and they appear in the wrong order upon start)
- If the user gives an invalid path jarvis throws an unhandled exception and the user dosen't get the chance to give another path.
- Could add more specific and detailed documentation

I would like to work on theese issues because I really like this project and I would love to contribute if it's ok with you. Let me know. Cheers!

pnhofmann commented on 11 Apr — Collaborator

Sure!

PR for bulkresizer plugin:



gianniskokkosis commented 24 days ago Contributor

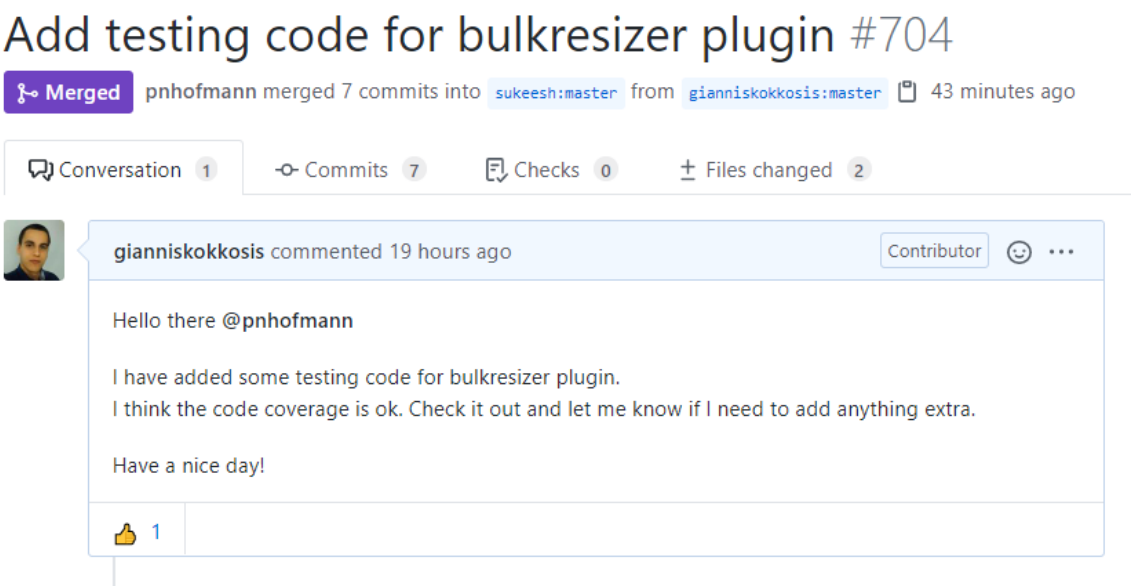
Hello!
I have finished some changes on bulkresizer plugin. This PR works on issue #676 . Please review the changes and let me know if there are any problems or issues that I need to fix **@pnhofmann** .

Changes made:

- Add colors on Jarvis' speech in order to look nicer.
- Split the code to smaller and more specific functions. This change aims on making the functions of the code more specific in order to execute a certain functionality and not multiple functionalities.
- Add functions that help the user to retype a path that leads to a valid dir, if she/he gives a path that lead to an invalid or not an existing dir
- Add a detailed documentation to all code's functions
- Add new capability to bulkresize plugin that creates an output path with the resized images if the path dosen't exist

P.S. I would like to add and some testing code as well! Tell me if it's ok with you.

Testing PR for bulkresizer plugin:



Add testing code for bulkresizer plugin #704

Merged **pnhofmann** merged 7 commits into `sukeesh:master` from `gianniskokkosis:master` 43 minutes ago

Conversation 1 Commits 7 Checks 0 Files changed 2

gianniskokkosis commented 19 hours ago Contributor

Hello there **@pnhofmann**

I have added some testing code for bulkresizer plugin.
I think the code coverage is ok. Check it out and let me know if I need to add anything extra.

Have a nice day!

1

A commit message for bulkresizer plugin:

Update bulkresize.py file

Browse files

```
- Add rename_img and output_path_concat functions

- This additions will seperate the login of renaming or the path concatination of the resized images from the resizing logic

- Now bulk_resizer function has only one functionality which is the resizing of the images
```

master (#692)

gianniskokkosis committed 29 days ago 1 parent 3721973 commit b0ee062f0056819e906599120494ed5a87b74a46

Issue for create_plugin.py plugin:

KonstantinosLolos commented on 7 Apr Contributor

Hello! I have been toying around with Jarvis on my computer and I would really like to contribute.
I was messing around with some ideas and I thought about adding a plugin that creates a new_plugin.py file with the Hello World plugin sample you provide and some extra documentation about the different options a developer has.
You guys have created really helpful info on how to contribute and create new plugins and since this is a plugin based System I thought it would be nice to apply the info you provide to a template! Any thoughts?

pnhofmann commented on 8 Apr Collaborator

Sounds good to me!

First PR for create_plugin.py plugin:

Add create_plugin.py plugin #684

Merged **pnhofmann** merged 7 commits into `sukeesh:master` from `KonstantinosLolos:master` on 13 May

Conversation 1 Commits 7 Checks 0 Files changed 1

KonstantinosLolos commented on 30 Apr Contributor

This plugin is designed to make it easier for a developer to start programming and experimenting with Jarvis plugins. It is functional for MAC and LINUX operating systems.
The format of the template which is created is strictly based on the instructions given in the url attached in README.md.

PS: if you decide to merge, might i suggest to mention the plugin in the README file under plugins compartment.
#674

Second PR for create_plugin.py plugin:

Merged pnhofmann merged 2 commits into `sukeesh:master` from `KonstantinosLolos:master` on 13 May

Conversation 1 | Commits 2 | Checks 0 | Files changed 6

KonstantinosLolos commented on 13 May — Contributor

Made some changes to the create_pluging.py file and some other files that made travis build fail because of PEP8 violations. #674

KonstantinosL added 2 commits on 13 May

- `Fix code to match PEP8 compliance` — `80f93a4`
- `Fix code in various file to match PEP8 guidelines` — `017c57d`

pnhofmann merged commit `9d89f70` into `sukeesh:master` on 13 May — Revert

pnhofmann commented on 13 May — Collaborator

Thanks ;)

Testing PR for create_plugin.py plugin:

KonstantinosLolos commented 26 days ago — Contributor

In this pull request:

- I have added some test cases for the create_plugin.py
- created an extra function to make the code cleaner.
- Fixed minor "too many blank lines issue in optional.py

I want also to add test cases for the plugin itself but I am having some trouble with mocking (I am now getting into it) and I cant seem to test the plugin without creating a file (as the plugin's utility is). Any tips as to where to look for info?
Thanks!
#674

KonstantinosL added 4 commits on 15 May

- `Add file_exists method to create_plugin.py plugin` — `1d9cdce`
- `Fix PEP8 violation in create_plugin.py` — `4f6a7fd`
- `Fix 'too many blank lines issue' in optional.py` — `772fdb5`
- `Add test cases for create_plugin.py` ··· — `6b417d3`

Issue for queue_input bug:

queue_input method works more like a stack rather than a queue #696

Closed **KonstantinosLolos** opened this issue 21 days ago · 3 comments

KonstantinosLolos commented 21 days ago • edited Contributor

Hi!
While working on some testing code with my colleague **@gianniskokkosis** We discovered that the queue_input method for plugin testing works in the logic of LIFO and thought that it would probably more reasonable to be called stack_input or something like that (+ it would clear out the logic of it a bit more).
Wondering if you have any idea on how this could change without having to replace everything manually.
It's something really minor but thought it could help in the comprehension of the method.
Opinion?
not really an issue, more like a thought

pnhofmann commented 21 days ago Collaborator

Nice catch ;).

I think, just like the name suggests, FIFO-Queue is probably what you'd expect of such a method. And what I had in mind.
So yes, this is a bug.
A bug that is unnoticed, because there just isn't a test, where order makes any difference. Let's fix the behavior of queue_input.

PR for queue_input bug:

Fix bug of _input_queue attribute acting like a stack #697

Merged **pnhofmann** merged 1 commit into `sukeesh:master` from `KonstantinosLolos:master` 20 days ago

KonstantinosLolos commented 20 days ago • edited Contributor

Fix bug of _input_queue attribute in function queue_input at **init**.py acting like a stack
Utilizes the deque module from collections library to change the
behaviour from LIFO to FIFO as it should #696

`Fix bug of _input_queue atribute in funcion tests/__init__.py acting …` f840758

A commit message for create_plugin.py plugin:

Remove path finding lines of code from inside the functions

Prev Next

```
since the path finding step is common for both plugin versions(macoz,linux) it is moved
out so both versions have access and avoid duplication.
(also some documentation rearrangement)
```

KonstantinosL committed on 30 Apr

commit dd67fd7803b2954a35949ef61151272167da6964

A commit message for queue_input:

Fix bug of _input_queue atribute in funcion tests/__init__.py acting ...

```
...like a stack

Utilizes the deque module from collections library to change the
behaviour from LIFO to FIFO as it should #696
```

KonstantinosL committed 20 days ago

commit f84075834b3dbcff48068c243ed957ddc33c3ee5

Reviews for the code and the PRs

While making a PR on Jarvis you do not have the ability to add reviewers to review your code. However we thought that it would be a good idea to tag the main collaborator to our PRs, in order to “solve” this problem.

Add testing code for bulkresizer plugin #704

Merged pnhofmann merged 7 commits into `sukeesh:master` from `gianniskokkosis:master` 43 minutes ago

Conversation 1 Commits 7 Checks 0 Files changed 2

gianniskokkosis commented 19 hours ago Contributor

Hello there @pnhofmann

I have added some testing code for bulkresizer plugin.
I think the code coverage is ok. Check it out and let me know if I need to add anything extra.

Have a nice day!

1