

BCED321 Advanced Programming

Assessment Two

Practical Assessment 2

Students:

Matthew Gordon

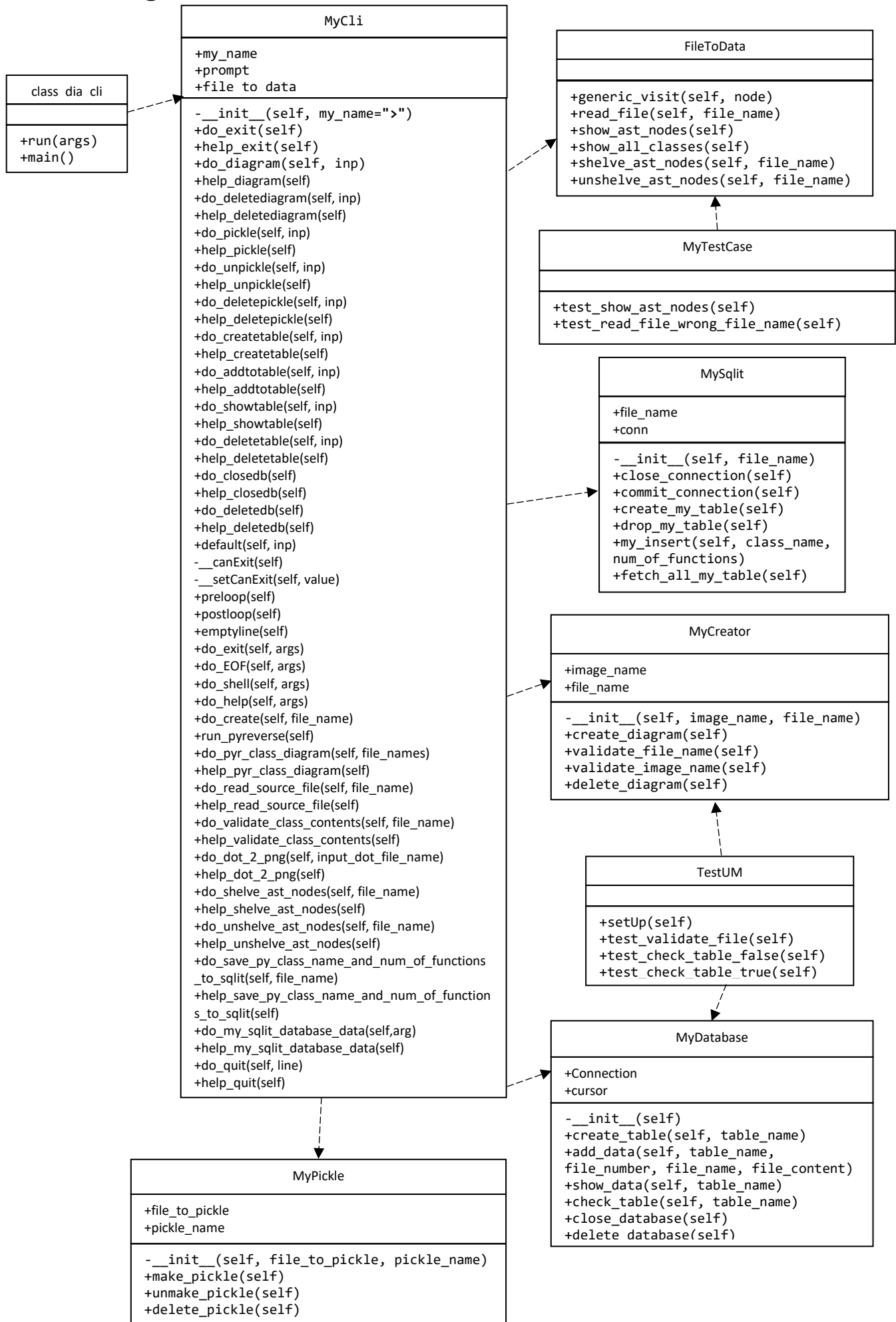
John Quiamco

Harry Lo

Table of Contents

1	Class Diagram	2
2	Matthew's help file details.....	3
3	John's help file details	3
4	Harry's help file details	3
5	Lists of Matthew's own work, self-reflection on robustness, and self-reflection on the completeness and implement.....	4
6	List of John's own work, self-reflection on robustness, and self-reflection on the completeness and implement.....	6
7	List of Harry's own work, self-reflection on robustness, and self-reflection on the completeness and implement.....	6
8	Location of GitHub repository.....	27

1 Class Diagram



2 Matthew's help file details

Command	Help
Diagram [inp]	Create a class diagram. Enter file location of py/dot file, then enter name/type of image.
Deletediagram [inp]	Deletes a diagram
Pickle [inp]	pickle [filename], enter file to pickle then the name of the pickle file
Unpickle [inp]	unpickle [picklefilename], enter the name of a text file that has been pickled
Deletepickle [inp]	Deletes a pickle file
Createtable [inp]	createtable [TABLE_NAME], creates a table with: file_number INTEGER PRIMARY KEY, file_name VARCHAR(30), 'file_content VARCHAR(999)
Addtotable [inp]	addtotable [TABLE_NAME], adds data to specified table
Showtable [inp]	showtable [TABLE_NAME], shows data held within specified table
Deletetable [inp]	deletetable [TABLE_NAME], deletes specified table
closedb	Closes current open database
deletedb	Deletes current database
default	Appears when an incorrect command is input

3 John's help file details

4 Harry's help file details

- 1) `python class_dia_cli.py --help` (in the system command line)

usage: class_dia_cli.py [-h] [-l LETTER]

This is a program going to a CLI to generate UML class diagram from Source

Codes

optional arguments:

-h, --help show this help message and exit

-l LETTER optional: give a letter displaced at the beginning of each command line. If user enter a string, only first character will be shown.

- 2) `>>>> help pyr_class_diagram` (in the line-oriented command interpreter)

Generate and display a class diagram in png format from a given python file

Syntax: `pyr_class_diagram [output png file name suffix] [input source code file name.py]`

- 3) `>>>> help read_source_file` (in the line-oriented command interpreter)

Extract data from the given python file to be an ast node

Syntax: `read_source_file [input source code file name.py]`

- 4) `>>>> help validate_class_contents` (in the line-oriented command interpreter)

Validate, list and display class names, function names and the total numbers of them in the given python file.

Class and function names are displayed in command line.

Total numbers of classes and functions are displayed in a bar graph.

Syntax: validate_class_contents [input source code file name.py].

5) >>>> help dot_2_png (in the line-oriented command interpreter)

Generate and display png file from the given dot file.

Syntax: dot_2_png [input dot file name.dot].

6) >>>> help shelve_ast_nodes (in the line-oriented command interpreter)

This function extracts data from the given python file to be an ast node and stores the node in files using shelve.

The files are given_file_name.py.db.bak, given_file_name.py.db.dat and given_file_name.py.db.dir.

The given file name should be [py_file_name.py]. The node will display as an indication of shelve done

Syntax: shelve_ast_nodes [input source code file name.py].

7) >>>> help unshelve_ast_nodes (in the line-oriented command interpreter)

This function retrieves data from the given shelved db file which stored an ast node by using shelve_ast_nodes command.

The given file name should have three corresponding files stored in the current directory.

The three files are given_file_name.py.db.bak, given_file_name.py.db.dat and given_file_name.py.db.dir.

The given file name should be [a_name.py.db]. The node will display as an indication of unshelve done

Syntax: unshelve_ast_nodes [a_name.py.db].

8) >>>> help quit (in the line-oriented command interpreter)

Quit from this CLI

:return: True

5 Lists of Matthew's own work, self-reflection on robustness, and self-reflection on the completeness and implement

	Component	Location	Used by your peers (2 mark)	Robustness (2 mark)	Complete and well implemented, i.e., "What is clever about this?" (2 mark)	Marks
1	Support command-line arguments	Lines 27-134 in my_cli.py	2 marks	2 marks. Encounters no unhandled exceptions	2 marks. Code follows pep8 guidelines and is Pythonic	6
2	Has a line-oriented command interpreter	Lines 27-134 in my_cli.py	2 marks	2 marks. Encounters no unhandled exceptions	2 marks. Code follows pep8 guidelines and is Pythonic	6

	based on cmd or similar package					
3	Display command line help of available commands	Lines 27-134 in my_cli.py	2 marks	2 marks. Encounters no unhandled exceptions	2 marks. All commands have a help command associated with them. Code follows pep8 guidelines and is Pythonic.	6
4	Change commands and options	Lines 27-134 in my_cli.py	2 marks	2 marks. Encounters no unhandled exceptions	2 marks. All commands have a help command associated with them that can be accessed by typing ? [COMMAND]. Code follows pep8 guidelines and is Pythonic	6
5	Extract data	Lines 11-15 in PickleMaker.py and Line 14 in DiagramCreator.py	2 marks	2 marks. Encounters no unhandled exceptions	2 marks. Data is extracted using Pickle. Code follows pep8 guidelines and is Pythonic	6
6	Validate data	Lines 88-94 in my_cli.py and 17-24 in DiagramCreator.py	2 marks	2 marks. Encounters no unhandled exceptions	2 marks. Code follows pep8 guidelines and is Pythonic	6
7	Provides object persistence / object serialization using either pickle or shelve	PickleMaker.py	2 marks	2 marks. Encounters no unhandled exceptions	2 marks. Code follows pep8 guidelines and is Pythonic	6
8	Can load data from a file	Lines 11-15 in PickleMaker.py and Lines 11 to 15 in DiagramCreator.py	2 marks	2 marks. Encounters no unhandled exceptions	2 marks. Code follows pep8 guidelines and is Pythonic	6
9	Can deal with file directory	PickleMaker.py and DiagramCreator.py	2 marks	1 mark. Some errors when creating a diagram in DiagramCreator.py due to it forcing a "classes." Prefix.	2 marks. Code follows pep8 guidelines and is Pythonic	5
10	Can raise exceptions and provide exception handling	Lines 88-94 in my_cli.py and 17-24 in DiagramCreator.py	2 marks	2 marks. Encounters no unhandled exceptions	2 marks. Code follows pep8 guidelines and is Pythonic	6
11	Amount of checking for	Lines 17-24 in DiagramCreator.py	2 marks	2 marks. Encounters no	2 marks. Function in SQLiteDatabase.py	6

	pre- and post-conditions of methods	and 63-69 in SQLDatabase.py		unhandled exceptions	checks that the table exists before allowing the code to process. Code in DiagramCreator.py checks that files exist before processing. Code follows pep8 guidelines and is Pythonic	
12	Provide doctests	MattDoctests.py	2 marks	2 marks. All tests pass	2 marks. 21 tests total.	6
13	Provide unittests	MattUnittests.py	2 marks	1 mark. All tests pass, but only 3 tests	1 mark. Only 3 tests	4
14	Pretty print, i.e., displaying data in chart/diagram, e.g., bar chart, pie chart, UML diagram, etc	DiagramCreator.py	2 marks	2 marks. Encounters no unhandled exceptions	2 marks. Code creates a class diagram. Code follows pep8 guidelines and is Pythonic	6
15	Can save and read data from a database, e.g., SQLite, MySQL and MongoDB	SQLDatabase.py	2 marks	2 marks. Encounters no unhandled exceptions	2 marks. Code follows pep8 guidelines and is Pythonic	6

6 List of John's own work, self-reflection on robustness, and self-reflection on the completeness and implement

7 List of Harry's own work, self-reflection on robustness, and self-reflection on the completeness and implement

1. Support command-line arguments

1.1. Used by peers

- File: class_dia_cli.py. I did the three functions below:
 - def run(args):
 - def main():
 - if __name__ == '__main__': (note that: this is an entry point of the whole program)

1.2. Robustness

- If user inputs wrong flag, my program will tell the user that the input was wrong as show below

```
C:\Users\harry\Documents\BCDE321Ass2>python class_dia_cli.py -s
usage: class_dia_cli.py [-h] [-l LETTER]
class_dia_cli.py: error: unrecognized arguments: -s
```

- There is exception handling as shown below. If there are any errors, the program will ask the user try again.

```
7 # Harry's work
8 def run(args):
9     my_cli = MyCli(args.letter[0])
10    try:
11        my_cli.cmdloop()
12    except Exception as err:
13        print("Please try again! The exception is: ", err)
```

1.3. Complete and well implemented

- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.
 - There are two blank lines between functions.

2. Has a line-oriented command interpreter based on cmd or similar package

2.1. Used by peers

- File: my_cli.py. I did the functions below:
 - def __init__(self, my_name=">"):
 - def do_pyr_class_diagram(self, file_names):
 - def help_pyr_class_diagram(self):
 - def do_read_source_file(self, file_name):
 - def help_read_source_file(self):
 - def do_validate_class_contents(self, file_name):
 - def help_validate_class_contents(self):
 - def do_dot_2_png(self, input_dot_file_name):
 - def help_dot_2_png(self):
 - def do_quit(self, line):
 - def help_quit(self):
 - if __name__ == '__main__': (for manual testing only)
- File: file_to_data.py. It is used by my_cli.py. I did the functions below:
 - def generic_visit(self, node):
 - def read_file(self, file_name):
 - def show_ast_nodes(self):
 - def show_all_classes(self):
 - if __name__ == "__main__": (for doctests and manual testing)

2.2. Robustness

- If the file, which user inputs into “def do_pyr_class_diagram(self, file_names):” function, does not exist, my program will tell the user that Your given python file does not exist in the current directory or your input arguments were wrong. The input arguments should be [png_file_name_suffix py_file_name.py]. Please try again! The screenshot is shown below:

```
>>>> pyr_class_diagram diagram tes
Your given python file does not exist in the current directory or you
>>>> |
```

- There is exception handling as shown below. If there are any errors, the program will ask the user try again.

```

19 # Harry's work
20 def do_pyr_class_diagram(self, file_names):
21     """Generate a class diagram in png format from given [png_file_name_suffix py_file_name.py]"""
22     self.file_names = file_names
23     python_file_name = file_names[(file_names.find(" ") + 1):]
24     try:
25         if path.exists(python_file_name):
26             pyreverse_command = 'pyreverse -ASmn -o png -p ' + file_names
27             subprocess.call(pyreverse_command)
28             print(file_names + ' are done')
29         else:
30             print("Your given python file does not exist in the current directory "
31                   "or your input arguments were wrong. The input arguments "
32                   "should be [png_file_name_suffix py_file_name.py]. |
33                   "Please try again!")
34     except Exception as err:
35         print("Please try again! The exception is: ", err)
36

```

2.3. Complete and well implemented

- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.
 - There are either one or two blank lines between code blocks according to PEP8.

3. Display command line help of available commands

3.1. Used by peers

- File: class_dia_cli.py and my_cli.py. Both files have the command line help as shown below:
 - For class_dia_cli.py, an example of the help function is below:

```

C:\Users\harry\Documents\BCDE321Ass2>python class_dia_cli.py -h
usage: class_dia_cli.py [-h] [-l LETTER]

This is a program going to a CLI to generate UML class diagram from Source
Codes

optional arguments:
  -h, --help  show this help message and exit
  -l LETTER  optional: give a letter displaced at the beginning of each
              command line. If user enter a string, only first character will
              be shown.

C:\Users\harry\Documents\BCDE321Ass2>

```

- For my_cli.py, two examples of the help functions are below:


```

>>>>> help

Documented commands (type help <topic>):
=====
displayallclasses      help                quit
displaysourcefilenodes pyr_class_diagram readsourcefile

Undocumented commands:
=====
dot_2_png

>>>>> help pyr_class_diagram
Generate a class diagram in png format from a given python file
Syntax: pyr_class_diagram [output png file name suffix] [input source code file name.py])
>>>>> █

```

3.2. Robustness

- File: class_dia_cli.py and my_cli.py. They both have exception handling as shown below.
 - For class_dia_cli.py, my program will tell the user to use -h or -help for help if the user used a wrong flag for help as shown below:

```

C:\Users\harry\Documents\BCDE321Ass2>python class_dia_cli.py -hela
usage: class_dia_cli.py [-h] [-l LETTER]
class_dia_cli.py: error: argument -h/--help: ignored explicit argument 'ela'

```

- For my_cli.py, my program will tell the user that no help on the command which does not exist or was wrongly spelled as shown below:

```

>>>>> help pyr_class_diagra
*** No help on pyr_class_diagra
>>>>> help wrong_command
*** No help on wrong_command
>>>>> █

```

3.3. Complete and well implemented

- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.
 - There are either one or two blank lines between code blocks according to PEP8.

4. Change commands and options

4.1. Used by peers

- File: my_cli.py. It can change options as shown below:
 - There are three options: (i) -h or -help flap for help; (ii) -l flap for adding a letter at the prompt as shown below (e.g. giving -l v flag will get the prompt of >>v>>); (iii) no flap for having a >>>>> prompt)

```

C:\Users\harry\Documents\BCDE321Ass2>python class_dia_cli.py -h
usage: class_dia_cli.py [-h] [-l LETTER]

This is a program going to a CLI to generate UML class diagram from Source
Codes

optional arguments:
  -h, --help  show this help message and exit
  -l LETTER  optional: give a letter displaced at the beginning of each
             command line. If user enter a string, only first character will
             be shown.

C:\Users\harry\Documents\BCDE321Ass2>python class_dia_cli.py -l v
>>> quit
Quitting.....

C:\Users\harry\Documents\BCDE321Ass2>python class_dia_cli.py
>>>> █

```

- For my_cli.py, there are more than one commands. An example of change commands (pyr_class_diagram and read_source_file commons) is below:

```

C:\Users\harry\Documents\BCDE321Ass2>python class_dia_cli.py
>>>> pyr_class_diagram trial test.py
parsing test.py...
trial test.py are done
>>>> read_source_file test.py
The ast nodes below has been read from the given python file, test.py:
Module(body=[ClassDef(name='Car', bases=[], keywords=[], body=[FunctionDef(
one. kwonlyvars=[]. kw defaults=[]. kwargs=None. defaults=[]). body=[Assign/

```

4.2. Robustness

- File: class_dia_cli.py and my_cli.py. They both have exception handling as shown below.
 - For class_dia_cli.py, my program will tell the user what options (i.e. flags) are available if the user used a wrong option (i.e. wrong flag) which is not available as shown below:

```

C:\Users\harry\Documents\BCDE321Ass2>python class_dia_cli.py -wrongFlag
usage: class_dia_cli.py [-h] [-l LETTER]
class_dia_cli.py: error: unrecognized arguments: -wrongFlag

```

- For my_cli.py, my program will tell the user if the user used a wrong function or a wrong argument as shown below:

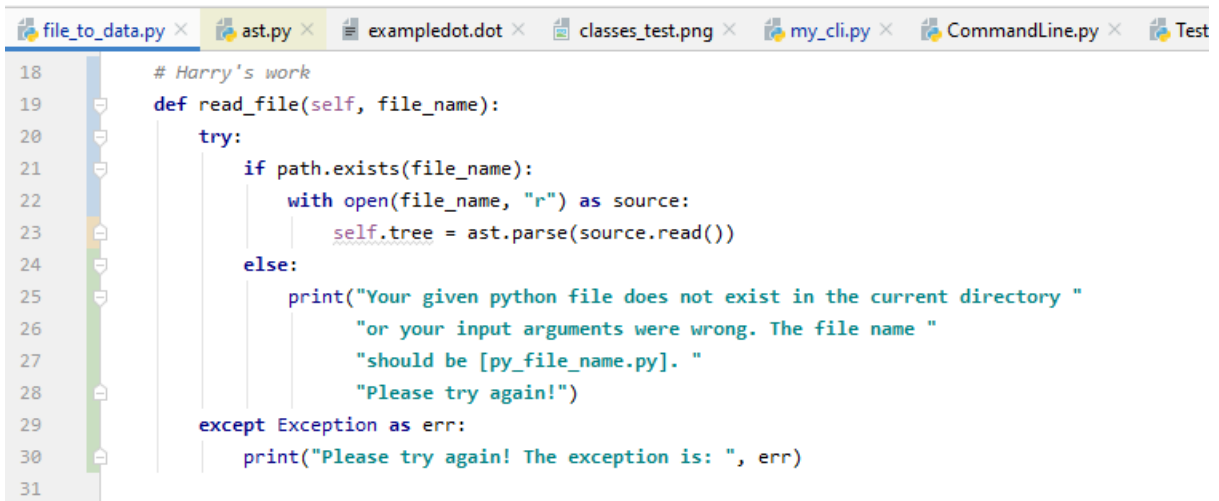
```
C:\Users\harry\Documents\BCDE321Ass2>python class_dia_cli.py
>>>> read_source_fil
*** Unknown syntax: read_source_fil
>>>> read_source_file
Your given python file does not exist in the current directory
or your input arguments were wrong. The input arguments
should be [py_file_name.py].
Please try again!
>>>> read_source_file te.py
Your given python file does not exist in the current directory
or your input arguments were wrong. The input arguments
should be [py_file_name.py].
Please try again!
>>>> █
```

4.3. Complete and well implemented

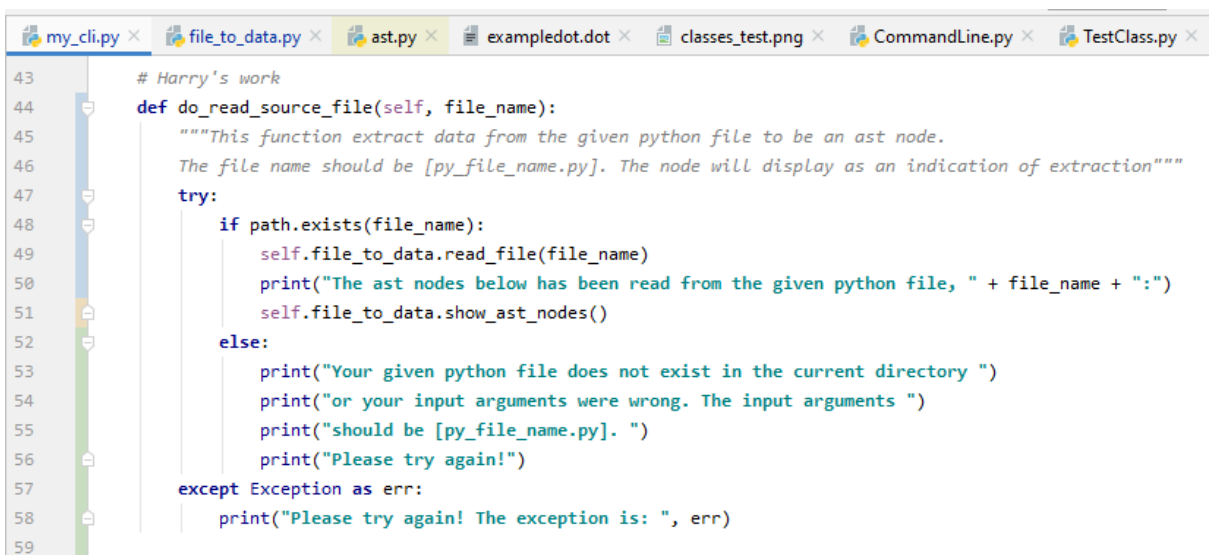
- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.
 - There are either one or two blank lines between code blocks according to PEP8.

5. Extract data

5.1. Data can be extracted from a python file through the def read_file(self, file_name): function in class FileToData(ast.NodeVisitor): in file_to_data.py. This read_file function is used by few functions in the my_cli.py, for example def do_read_source_file(self, file_name):. The codes of the two functions are shown below:



```
file_to_data.py x ast.py x exampledot.dot x classes_test.png x my_cli.py x CommandLine.py x Test
18 # Harry's work
19 def read_file(self, file_name):
20     try:
21         if path.exists(file_name):
22             with open(file_name, "r") as source:
23                 self.tree = ast.parse(source.read())
24         else:
25             print("Your given python file does not exist in the current directory "
26                 "or your input arguments were wrong. The file name "
27                 "should be [py_file_name.py]. "
28                 "Please try again!")
29     except Exception as err:
30         print("Please try again! The exception is: ", err)
31
```



```
my_cli.py x file_to_data.py x ast.py x exampledot.dot x classes_test.png x CommandLine.py x TestClass.py x
43 # Harry's work
44 def do_read_source_file(self, file_name):
45     """This function extract data from the given python file to be an ast node.
46     The file name should be [py_file_name.py]. The node will display as an indication of extraction"""
47     try:
48         if path.exists(file_name):
49             self.file_to_data.read_file(file_name)
50             print("The ast nodes below has been read from the given python file, " + file_name + ":")
51             self.file_to_data.show_ast_nodes()
52         else:
53             print("Your given python file does not exist in the current directory ")
54             print("or your input arguments were wrong. The input arguments ")
55             print("should be [py_file_name.py]. ")
56             print("Please try again!")
57     except Exception as err:
58         print("Please try again! The exception is: ", err)
59
```

5.2. Robustness

- Both aforementioned `def read_file(self, file_name):` and `def do_read_source_file(self, file_name):` functions have exception handling which checks if the file exists or not and if there is error or not. My program will tell the users if file does not exist in current directory or there are errors as shown in the codes at item 5.1 above.

5.3. Complete and well implemented

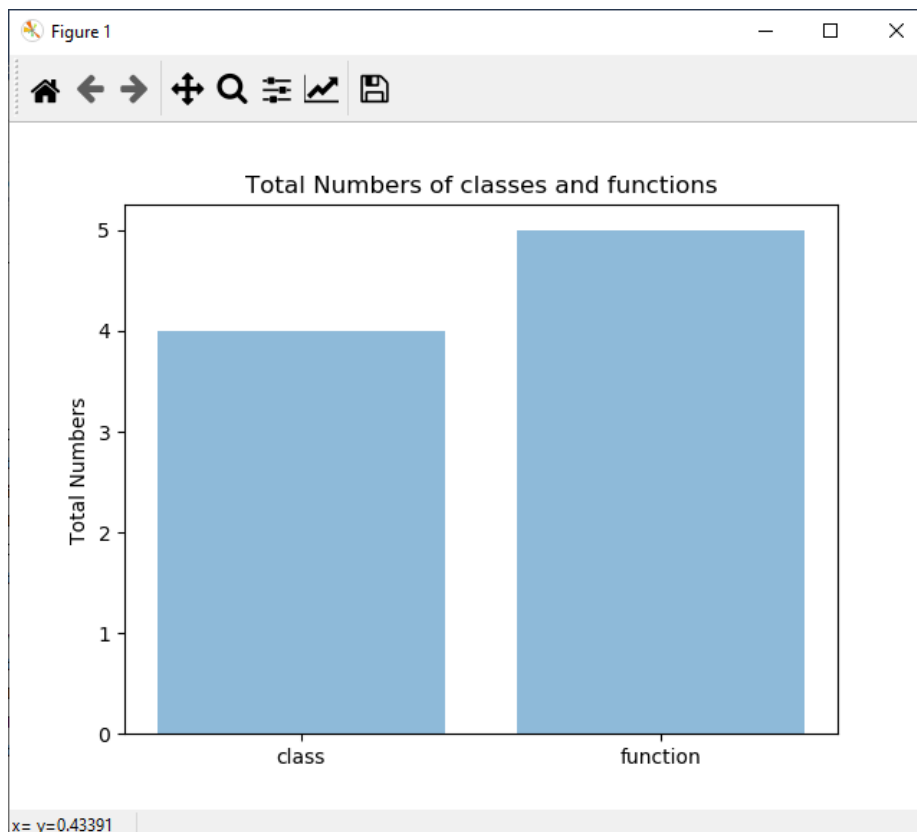
- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.
 - There are either one or two blank lines between code blocks according to PEP8.

6. Validate data

6.1. Used by peers

- File: `my_cli.py`. The `def do_validate_class_contents(self, file_name):` function validates class names, function names and the total numbers of them in the given python file, and display them in command lines and a graph as below:

```
C:\Users\harry\Documents\BCDE321Ass2>python class_dia_cli.py
>>>> validate_class_contents test.py
---There are 4 classes.-----
----The classes are: -----
-----Car class
-----Door class
-----Wheel class
-----Taxi class
-----The Car class has 2 functions
-----The functions in Car class are
-----__init__ function
-----is_sold function
-----The Door class has 1 functions
-----The functions in Door class are
-----__init__ function
-----The Wheel class has 1 functions
-----The functions in Wheel class are
-----__init__ function
-----The Taxi class has 1 functions
-----The functions in Taxi class are
-----__init__ function
total number of classes is 4
total number of functions is 5
```



6.2. Robustness

- The def do_validate_class_contents(self, file_name): function has exception handling which checks if the file exists or not and if there is error or not. My program will tell the users if file does not exist in current directory or there are errors as shown in the codes below. There are "try" and "if path.exists(file_name)".

```
t.py × my_cli.py × test.py × trial.py × file_to_data.py × class_dia_cli.py ×
# Harry's work
def do_validate_class_contents(self, file_name):
    """Validate, list and display class names, function names and the total numbers of them
    in the given python file. Class and function names are displayed in command line.
    Total numbers of classes and functions are displayed in a bar graph.
    Syntax: validate_class_contents [input source code file name.py]"""

    # sample: validate_class_contents test.py
    num_of_classes = 0
    num_of_functions = 0
    try:
        if path.exists(file_name):
            self.file_to_data.read_file(file_name)
            num_of_classes = len(self.file_to_data.tree.body)
            print("---There are " + str(num_of_classes) + " classes.-----")
            print("-----The classes are: -----")
            for my_class in self.file_to_data.tree.body:
                print("-----" + my_class.name + " class")
            for my_class in self.file_to_data.tree.body:
                print("-----The " + my_class.name + " class has " + str(len(my_class.body)) + " functions")
                num_of_functions += len(my_class.body)
                print("-----The functions in " + my_class.name + " class are ")
                for my_function in my_class.body:
                    print("-----" + my_function.name + " function")
            print("total number of classes is " + str(num_of_classes))
            print("total number of functions is " + str(num_of_functions))
            # for my_function in my_class.body:
```

6.3. Complete and well implemented

- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.

- There are either one or two blank lines between code blocks according to PEP8.

7. Provides object-persistence / object serialization using either pickle or shelve

7.1. Used by peers

- `shelve_ast_nodes` and `unshelve_ast_nodes` commands in `my_cli.py` file can be used by peers. The commands used shelve to store and retrieve an ast node in and from a specific db file.

7.2. Robustness

- Those commands call `def shelve_ast_nodes(self, file_name):` and `def unshelve_ast_nodes(self, file_name):` in `file_to_data.py`. They have exception handling as shown below:

```

56      # print(len(tree.body)) # show 4 classes
57
58      # Harry's work
59      def shelve_ast_nodes(self, file_name):
60          db_file_name = file_name + ".db"
61          try:
62              if path.exists(file_name):
63                  with open(file_name, "r") as source:
64                      self.tree = ast.parse(source.read())
65              try:
66                  shelve_tree = shelve.open(db_file_name)
67                  shelve_tree[db_file_name] = self.tree
68              except Exception as err:
69                  print("Please try again! The exception is: ", err)
70              finally:
71                  shelve_tree.close()
72          else:
73              print("Your given python file does not exist in the current directory "
74                    "or your input arguments were wrong. The file name "
75                    "should be [py_file_name.py]. "
76                    "Please try again!")
77          except Exception as err:
78              print("Please try again! The exception is: ", err)
79
80      # Harry's work
81      def unshelve_ast_nodes(self, file_name):
82          try:
83              unshelve_object = shelve.open(file_name)
84              self.tree = unshelve_object[file_name]
85          except Exception as err:
86              print("Please try again! The exception is: ", err)
87          finally:
88              unshelve_object.close()
89

```

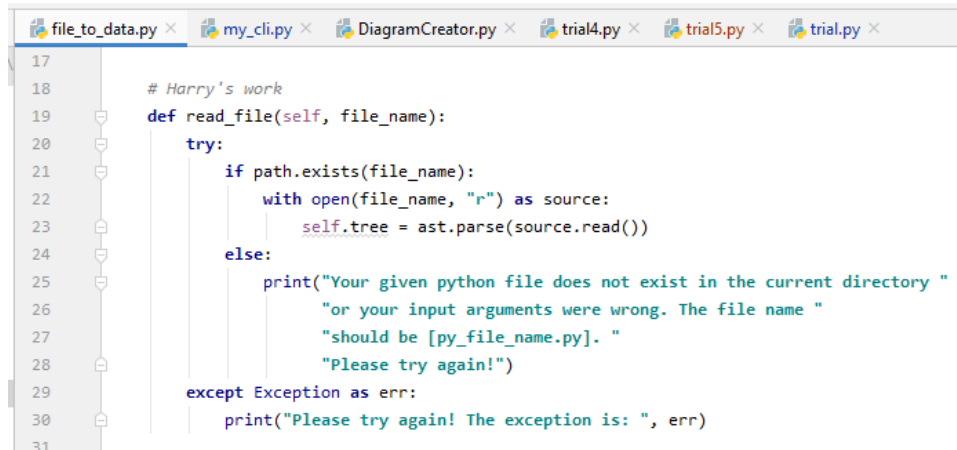
7.3. Complete and well implemented

- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.
 - There are either one or two blank lines between code blocks according to PEP8.

8. Can load data from a file

8.1. Used by peers

- File: file_to_data.py. Data can be loaded from a python file through the def read_file(self, file_name): function as shown below in class FileToData(ast.NodeVisitor): in file_to_data.py file. This read file function is used by do_read_source_file(self, file_name): function, def do_validate_class_contents(self, file_name): function, and def do_validate_class_contents(self, file_name): function in my_cli.py file.



```

17
18     # Harry's work
19     def read_file(self, file_name):
20         try:
21             if path.exists(file_name):
22                 with open(file_name, "r") as source:
23                     self.tree = ast.parse(source.read())
24             else:
25                 print("Your given python file does not exist in the current directory "
26                     "or your input arguments were wrong. The file name "
27                     "should be [py_file_name.py]. "
28                     "Please try again!")
29         except Exception as err:
30             print("Please try again! The exception is: ", err)
31

```

8.2. Robustness

- The aforementioned def read_file(self, file_name): have exception handling which checks if the file exists or not and if there is error or not. My program will tell the users if file does not exist in current directory or there are errors as shown in the codes at item 8.1 above.

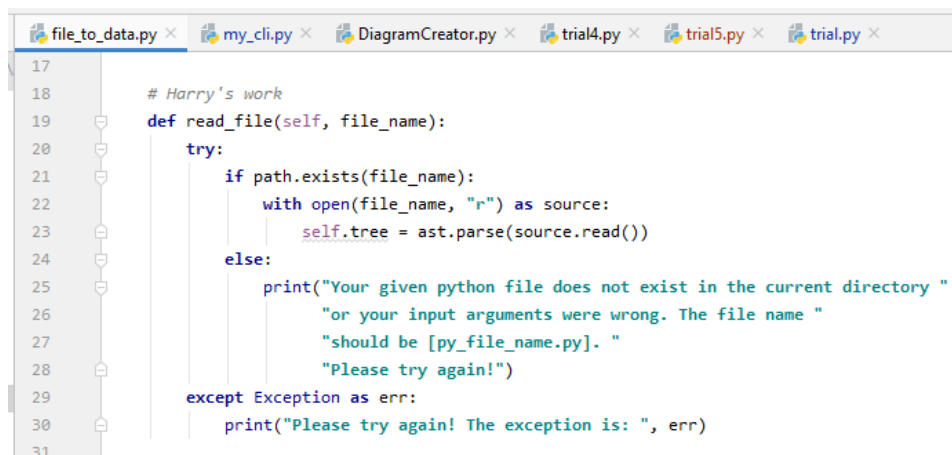
8.3. Complete and well implemented

- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.
 - There are either one or two blank lines between code blocks according to PEP8.

9. Can deal with file directory

9.1. Used by peers

- File: file_to_data.py. The path.exists() function is used in def read_file(self, file_name): function to check if the given file is in the current file directory or not. The program will tell the user if the file is not in the current file directory. The corresponding code is below:



```

17
18     # Harry's work
19     def read_file(self, file_name):
20         try:
21             if path.exists(file_name):
22                 with open(file_name, "r") as source:
23                     self.tree = ast.parse(source.read())
24             else:
25                 print("Your given python file does not exist in the current directory "
26                     "or your input arguments were wrong. The file name "
27                     "should be [py_file_name.py]. "
28                     "Please try again!")
29         except Exception as err:
30             print("Please try again! The exception is: ", err)
31

```

9.2. Robustness

- The aforementioned path.exists() function is for exception handling which checks if the file is in current file directory or not. My program will tell the users if the current directory does not have the file.

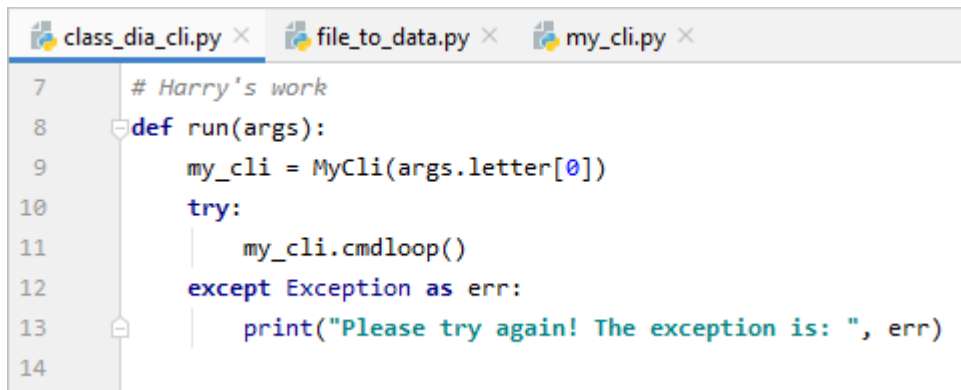
9.3. Complete and well implemented

- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.
 - There are either one or two blank lines between code blocks according to PEP8.

10. Can raise exceptions and provide exception handling

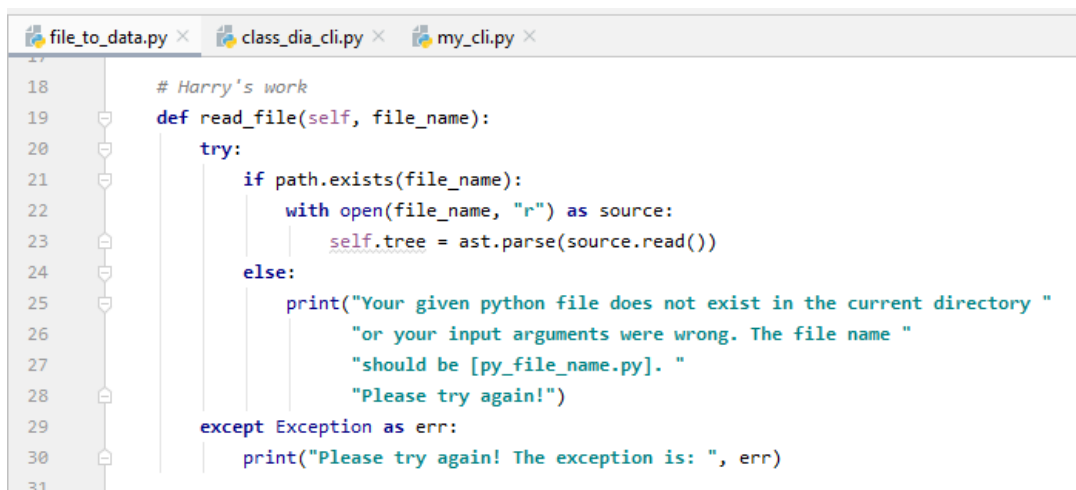
10.1. Used by peers

- I have provided exception handling in different parts of the codes. Few examples only (not all) are given below:
 - `def run(args):` function in `class_dia_cli.py` as shown below:



```
class_dia_cli.py x file_to_data.py x my_cli.py x
7      # Harry's work
8      def run(args):
9          my_cli = MyCli(args.letter[0])
10         try:
11             my_cli.cmdloop()
12         except Exception as err:
13             print("Please try again! The exception is: ", err)
14
```

- `def read_file(self, file_name):` function in `file_to_data.py` as shown below:



```
file_to_data.py x class_dia_cli.py x my_cli.py x
17
18     # Harry's work
19     def read_file(self, file_name):
20         try:
21             if path.exists(file_name):
22                 with open(file_name, "r") as source:
23                     self.tree = ast.parse(source.read())
24             else:
25                 print("Your given python file does not exist in the current directory "
26                       "or your input arguments were wrong. The file name "
27                       "should be [py_file_name.py]. "
28                       "Please try again!")
29         except Exception as err:
30             print("Please try again! The exception is: ", err)
31
```

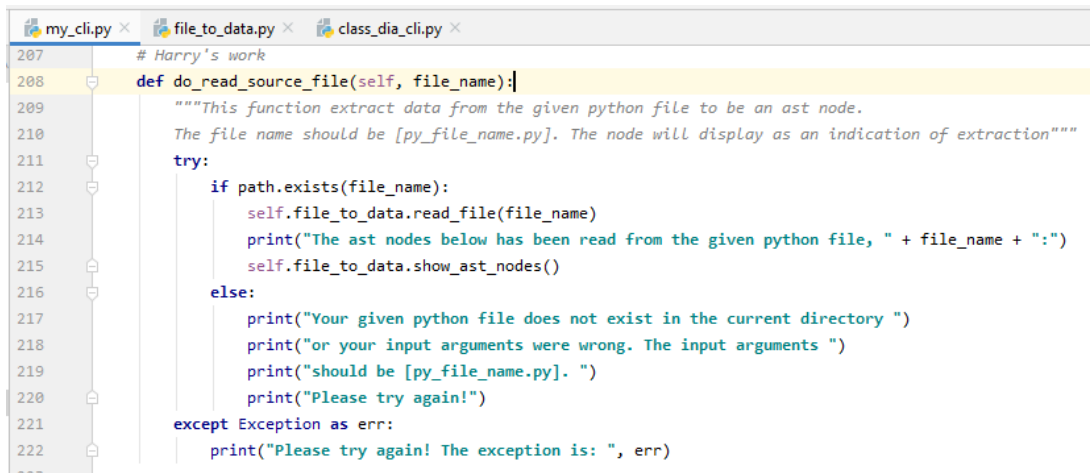
- `def do_pyr_class_diagram(self, file_names):` function in `my_cli.py` as shown below:



```
my_cli.py x file_to_data.py x class_dia_cli.py x
171     # Harry's work
172     def do_pyr_class_diagram(self, file_names):
173         """Generate and display a class diagram in png format from given [png_file_name_suffix py_file_name.py]"""
174         self.file_names = file_names
175         python_file_name = file_names[(file_names.find(" ") + 1):]
176         png_file_name = 'classes_' + file_names[0:(file_names.find(" ")) + 1] + '.png'
177         try:
178             if path.exists(python_file_name):
179                 pyreverse_command = 'pyreverse -ASmn -o png -p ' + file_names
180                 subprocess.call(pyreverse_command)
181                 print(file_names + ' are done')
182             if path.exists(png_file_name):
183                 # show png image
184                 img = mpimg.imread(png_file_name)
185                 fig = plt.imshow(img)
186                 fig.axes.get_xaxis().set_visible(False)
187                 fig.axes.get_yaxis().set_visible(False)
188                 plt.show()
189             else:
190                 print("The image of class diagram cannot be generate.")
191                 print("Please check with your system administrators.")
192         else:
193
```

MyCli > do_pyr_class_diagram()

- `def do_read_source_file(self, file_names):` function in `my_cli.py` as shown below:

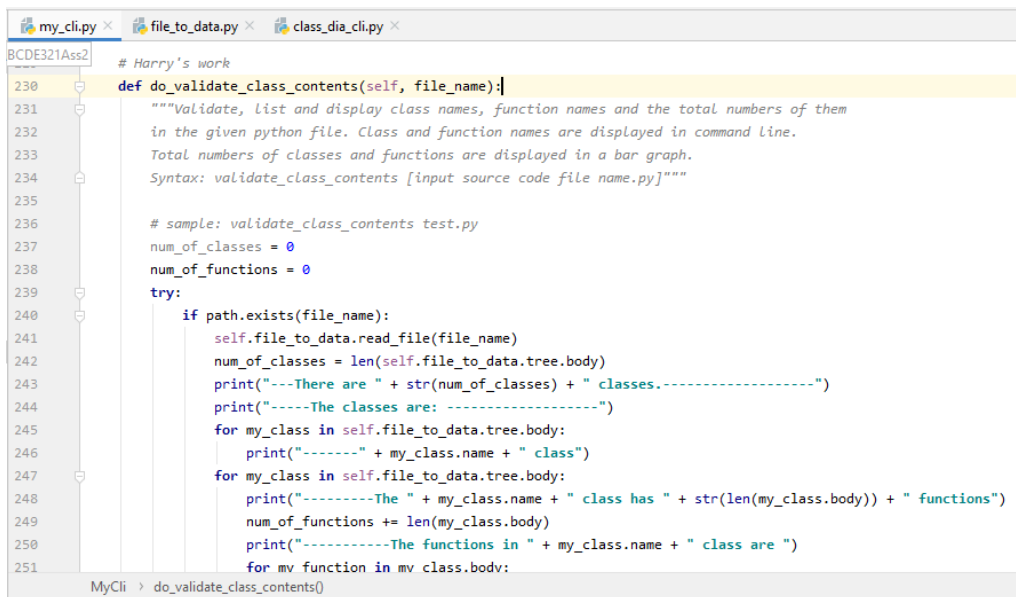


```

207 # Harry's work
208 def do_read_source_file(self, file_name):
209     """This function extract data from the given python file to be an ast node.
210     The file name should be [py_file_name.py]. The node will display as an indication of extraction"""
211     try:
212         if path.exists(file_name):
213             self.file_to_data.read_file(file_name)
214             print("The ast nodes below has been read from the given python file, " + file_name + ":")
215             self.file_to_data.show_ast_nodes()
216         else:
217             print("Your given python file does not exist in the current directory ")
218             print("or your input arguments were wrong. The input arguments ")
219             print("should be [py_file_name.py]. ")
220             print("Please try again!")
221     except Exception as err:
222         print("Please try again! The exception is: ", err)

```

- def do_validate_class_contents(self, file_name): function in my_cli.py as shown below:

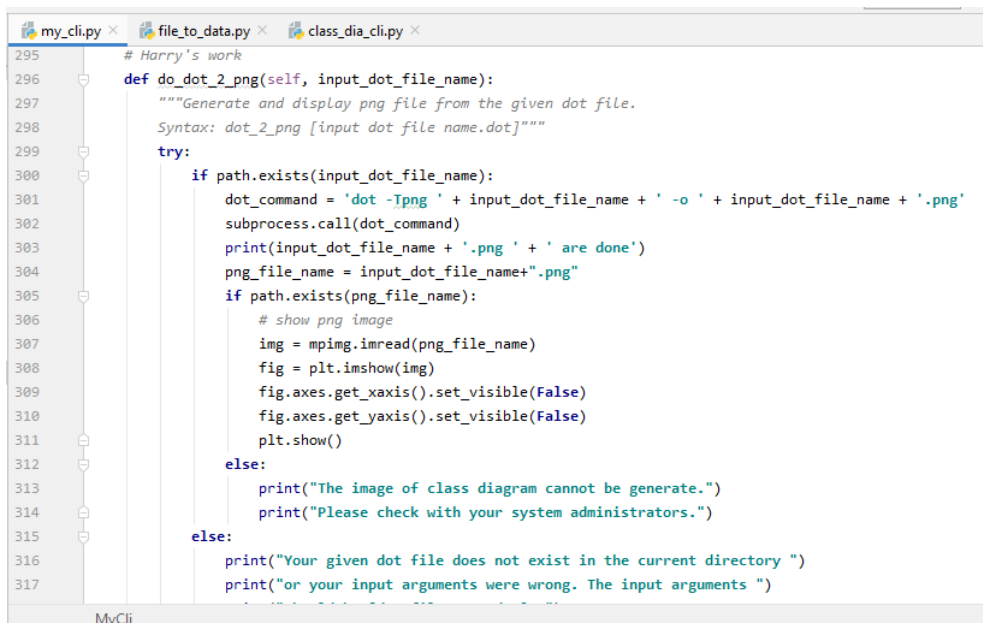


```

230 def do_validate_class_contents(self, file_name):
231     """Validate, List and display class names, function names and the total numbers of them
232     in the given python file. Class and function names are displayed in command line.
233     Total numbers of classes and functions are displayed in a bar graph.
234     Syntax: validate_class_contents [input source code file name.py]"""
235
236     # sample: validate_class_contents test.py
237     num_of_classes = 0
238     num_of_functions = 0
239     try:
240         if path.exists(file_name):
241             self.file_to_data.read_file(file_name)
242             num_of_classes = len(self.file_to_data.tree.body)
243             print("----There are " + str(num_of_classes) + " classes.-----")
244             print("----The classes are: -----")
245             for my_class in self.file_to_data.tree.body:
246                 print("-----" + my_class.name + " class")
247             for my_class in self.file_to_data.tree.body:
248                 print("-----The " + my_class.name + " class has " + str(len(my_class.body)) + " functions")
249                 num_of_functions += len(my_class.body)
250             print("-----The functions in " + my_class.name + " class are ")
251             for mv function in mv class.body:

```

- def do_dot_2_png(self, input_dot_file_name): function in my_cli.py as shown below:



```

295 # Harry's work
296 def do_dot_2_png(self, input_dot_file_name):
297     """Generate and display png file from the given dot file.
298     Syntax: dot_2_png [input dot file name.dot]"""
299     try:
300         if path.exists(input_dot_file_name):
301             dot_command = 'dot -Tpng ' + input_dot_file_name + ' -o ' + input_dot_file_name + '.png'
302             subprocess.call(dot_command)
303             print(input_dot_file_name + '.png ' + ' are done')
304             png_file_name = input_dot_file_name + ".png"
305             if path.exists(png_file_name):
306                 # show png image
307                 img = mpimg.imread(png_file_name)
308                 fig = plt.imshow(img)
309                 fig.axes.get_xaxis().set_visible(False)
310                 fig.axes.get_yaxis().set_visible(False)
311                 plt.show()
312             else:
313                 print("The image of class diagram cannot be generate.")
314                 print("Please check with your system administrators.")
315         else:
316             print("Your given dot file does not exist in the current directory ")
317             print("or your input arguments were wrong. The input arguments ")

```

10.2. Robustness

- The aforementioned functions have exception handling which checks if there is error or not. My program will tell the users if error.

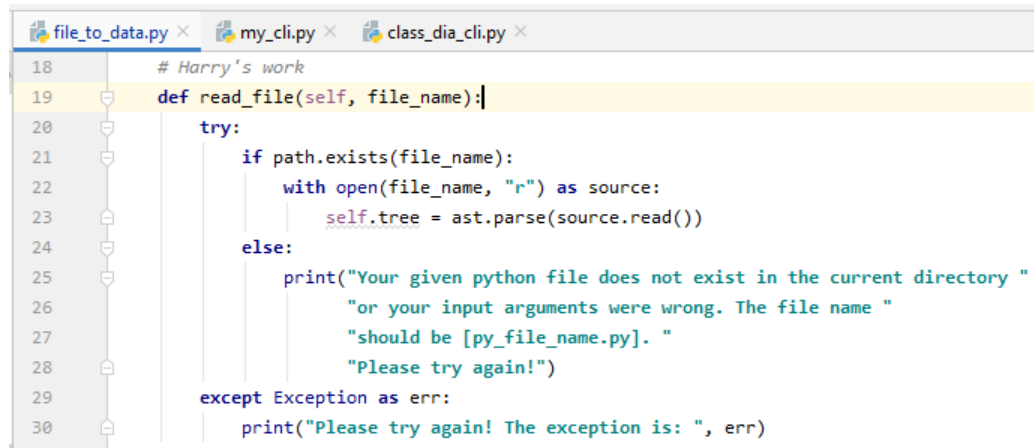
10.3. Complete and well implemented

- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.
 - There are either one or two blank lines between code blocks according to PEP8.

11. Amount of checking for pre- and post- conditions of methods

11.1. Used by peers

- Files: file_to_data.py and my_cli.py.
 - The def read_file(self, file_name): function as shown below in file_to_data.py has check pre-condition which checks if the required file exists in the current directory before the file is opened.

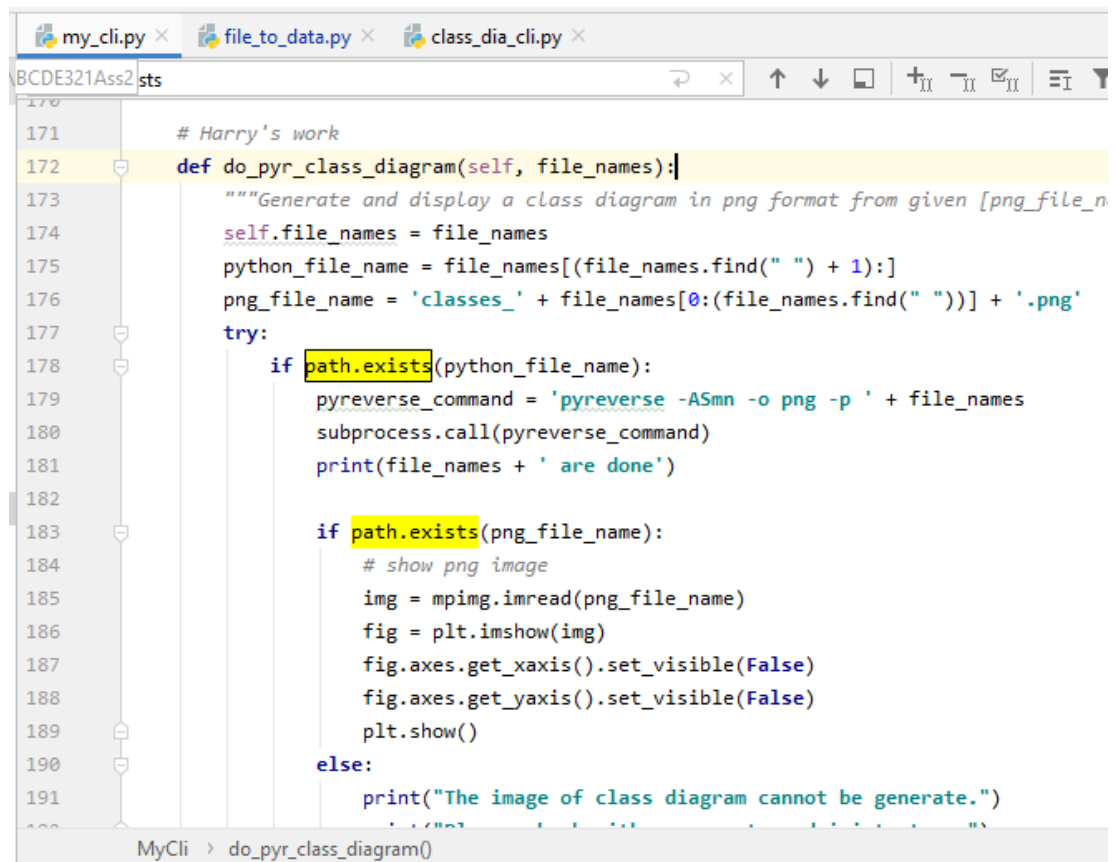


```

18 # Harry's work
19 def read_file(self, file_name):
20     try:
21         if path.exists(file_name):
22             with open(file_name, "r") as source:
23                 self.tree = ast.parse(source.read())
24         else:
25             print("Your given python file does not exist in the current directory "
26                 "or your input arguments were wrong. The file name "
27                 "should be [py_file_name.py]. "
28                 "Please try again!")
29     except Exception as err:
30         print("Please try again! The exception is: ", err)

```

- The def do_pyr_class_diagram(self, file_names): function as shown below in my_cli.py has check both pre- and post- conditions which checks if both the required input file and the output file respectively as show below exist in the current directory before the files are opened.



```

171 # Harry's work
172 def do_pyr_class_diagram(self, file_names):
173     """Generate and display a class diagram in png format from given [png_file_n
174     self.file_names = file_names
175     python_file_name = file_names[(file_names.find(" ") + 1):]
176     png_file_name = 'classes_' + file_names[0:(file_names.find(" "))] + '.png'
177     try:
178         if path.exists(python_file_name):
179             pyreverse_command = 'pyreverse -ASmn -o png -p ' + file_names
180             subprocess.call(pyreverse_command)
181             print(file_names + ' are done')
182
183         if path.exists(png_file_name):
184             # show png image
185             img = mpimg.imread(png_file_name)
186             fig = plt.imshow(img)
187             fig.axes.get_xaxis().set_visible(False)
188             fig.axes.get_yaxis().set_visible(False)
189             plt.show()
190         else:
191             print("The image of class diagram cannot be generate.")

```

- The def do_read_source_file(self, file_name): function as shown below in my_cli.py has check pre-condition which checks if the required file exists in the current directory before the file is opened.

```

207 # Harry's work
208 def do_read_source_file(self, file_name):
209     """This function extract data from the given python file to be an ast node.
210     The file name should be [py_file_name.py]. The node will display as an indication of extraction"""
211     try:
212         if path.exists(file_name):
213             self.file_to_data.read_file(file_name)
214             print("The ast nodes below has been read from the given python file, " + file_name + ":")
215             self.file_to_data.show_ast_nodes()
216         else:
217             print("Your given python file does not exist in the current directory ")
218             print("or your input arguments were wrong. The input arguments ")
219             print("should be [py_file_name.py]. ")
220             print("Please try again!")
221     except Exception as err:
222         print("Please try again! The exception is: ", err)

```

- The def do_validate_class_contents(self, file_name): function as shown below in my_cli.py has check pre-condition which checks if the required file exists in the current directory before the file is opened.

```

229 # Harry's work
230 def do_validate_class_contents(self, file_name):
231     """Validate, list and display class names, function names and the total numbers of them
232     in the given python file. Class and function names are displayed in command line.
233     Total numbers of classes and functions are displayed in a bar graph.
234     Syntax: validate_class_contents [input source code file name.py]"""
235
236     # sample: validate_class_contents test.py
237     num_of_classes = 0
238     num_of_functions = 0
239     try:
240         if path.exists(file_name):
241             self.file_to_data.read_file(file_name)
242             num_of_classes = len(self.file_to_data.tree.body)
243             print("----There are " + str(num_of_classes) + " classes.-----")
244             print("-----The classes are: -----")
245             for my_class in self.file_to_data.tree.body:
246                 print("-----" + my_class.name + " class")
247             for my_class in self.file_to_data.tree.body:
248                 print("-----The " + my_class.name + " class has " + str(len(my_class.body)) + " functions")
249                 num_of_functions += len(my_class.body)

```

- The def do_dot_2_png(self, input_dot_file_name): function as shown below in my_cli.py has check both pre- and post- conditions which checks if both the required input file and the output file respectively as show below exist in the current directory before the files are opened.

```

295 # Harry's work
296 def do_dot_2_png(self, input_dot_file_name):
297     """Generate and display png file from the given dot file.
298     Syntax: dot_2_png [input dot file name.dot]"""
299     try:
300         if path.exists(input_dot_file_name):
301             dot_command = 'dot -Tpng ' + input_dot_file_name + ' -o ' + input_dot_file_name + '.png'
302             subprocess.call(dot_command)
303             print(input_dot_file_name + '.png ' + ' are done')
304             png_file_name = input_dot_file_name + ".png"
305             if path.exists(png_file_name):
306                 # show png image
307                 img = mpimg.imread(png_file_name)
308                 fig = plt.imshow(img)
309                 fig.axes.get_xaxis().set_visible(False)
310                 fig.axes.get_yaxis().set_visible(False)
311                 plt.show()
312             else:
313                 print("The image of class diagram cannot be generate.")
314                 print("Please check with your system administrators.")
315         else:
316             print("Your given dot file does not exist in the current directory ")

```

11.2. Robustness

- The path.exists() function in the aforementioned codes is for both pre- and post- conditions (for input file and output file respectively) which checks if the file is in current file directory or not. My program will tell the users if the current directory does not have the file.

11.3. Complete and well implemented

- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.
 - There are either one or two blank lines between code blocks according to PEP8.

12. Provide doctests

12.1. Used by peers

- File: file_to_data.py
 - Two doctests are shown below:

```

10 # Harry's work
11 class FileToData(ast.NodeVisitor):
12     """doctest
13     >>> file_to_data = FileToData()
14     >>> file_to_data.read_file("test.py")
15     >>> file_to_data.show_ast_nodes()
16     Module(body=[ClassDef(name='Car', bases=[], keywords=[], body=[FunctionDef(name='__init__
17
18     >>> file_to_data = FileToData()
19     >>> file_to_data.read_file("test.p")
20     Your given python file does not exist in the current directory or your input arguments we
21

```

```

57
58 # Harry's work
59 if __name__ == "__main__":
60     # ---below is for manual testing only
61     # file_to_data = FileToData()
62     # file_to_data.read_file("test.p")
63     #file_to_data.show_ast_nodes()
64     # file_to_data.show_all_classes()
65     # ^^^below is for manual testing only
66     import doctest
67     doctest.testmod()
68

```

12.2. Robustness

- All doctests were passed as shown below

```

BCDE321Ass2 > file_to_data.py
Project
  BCDE321Ass2
    John files
    Matts files
      CommandLine.py
      DiagramCreator.py
      my_cli.py
      PickleMaker.py
      SQLDatabase.py
      TestClass.py
      .github
      .gitignore
      BCDE321AssDoc.docx
      class.png
      class_dia_cli.py
  file_to_data.py
    58 # Harry's work
    59 if __name__ == "__main__":
    60     # ---below is for manual testing only
    61     # file_to_data = FileToData()
    62     # file_to_data.read_file("test.p")
    63     #file_to_data.show_ast_nodes()
    64     # file_to_data.show_all_classes()
    65     # ^^^below is for manual testing only
    66
    67 # doctests
    68 import doctest
    69 doctest.testmod()
    70
    if __name__ == "__main__"

Terminal: Local
C:\Users\harry\Documents\BCDE321Ass2>python file_to_data.py
C:\Users\harry\Documents\BCDE321Ass2>

```

12.3. Complete and well implemented

- Less than 10 different doctests

13. Provide unittests

13.1. Used by peers

- File: test_file_to_data.py
 - Two unit tests are shown below:

```

test_file_to_data.py
1 import unittest
2 from file_to_data import FileToData
3 from io import StringIO
4 import sys
5
6
7 class MyTestCase(unittest.TestCase):
8     def test_show_ast_nodes(self):
9         captured_output = StringIO() # Create StringIO object
10        sys.stdout = captured_output # and redirect stdout.
11        file_to_data = FileToData()
12        file_to_data.read_file("test.py")
13        file_to_data.show_ast_nodes()
14        sys.stdout = sys.__stdout__
15        actual_output = captured_output.getvalue()
16        expected_output = "Module(body=[ClassDef(name='Car', bases=[], keywords=[], body=[FunctionDef(" \
17        "name='__init__', args=arguments(args=[arg(arg='self', annotation=None), " \
18        "arg(arg='num', annotation=None)], vararg=None, kwonlyargs=[], kw_defaults=[], " \
19        "kwarg=None, defaults=[], body=[Assign(targets=[Attribute(value=Name(id='self', " \
20        "ctx=Load()), attr='door', ctx=Store()), value=Call(func=Name(id='Door', ctx=Load()), " \
21        "args=[Num(n=1)], keywords=[]), Assign(targets=[Attribute(value=Name(id='self', " \
22        "ctx=Load()), attr='wheel', ctx=Store()), value=Call(func=Name(id='Wheel', ctx=Load()), " \
23        "args=[Num(n=1)], keywords=[])], decorator_list=[], returns=None), FunctionDef(" \
24        "name='is_sold', args=arguments(args=[arg(arg='self', annotation=None)], vararg=None, " \
25        "kwonlyargs=[], kw_defaults=[], body=[Expr(value=Call(func=Name " \
26        "(id='print', ctx=Load()), args=[Str(s='this car is sold')], keywords=[]), d" \
27        "ecorator_list=[], returns=None)], decorator_list=[], ClassDef(name='Door', " \
28        "
29
MyTestCase

```

```

42         "ctx=Load(), args=[], keywords=[]), Assign(targets=[Attribute(value=Name(id='self', " \
43         "ctx=Load(), attr='color', ctx=Store()), value=Str(s='red'))], decorator_list=[], " \
44         "returns=None)], decorator_list=[]))\n"
45
46     self.assertEqual(actual_output, expected_output)
47
48     def test_read_file_wrong_file_name(self):
49         captured_output = StringIO() # Create StringIO object
50         sys.stdout = captured_output # and redirect stdout.
51         file_to_data = FileToData()
52         file_to_data.read_file("test.p")
53         sys.stdout = sys.__stdout__
54         actual_output = captured_output.getvalue()
55         expected_output = "Your given python file does not exist in the current directory or your input " \
56         "arguments were wrong. The file name should be [py_file_name.py]. Please try again!\n"
57
58         actual_output = 0
59
60         expected_output = 0
61         self.assertEqual(actual_output, expected_output)
62
63
64     if __name__ == '__main__':
65         unittest.main()
66

```

MyTestCase > test_read_file_wrong_file_name()

13.2. Robustness

- All unit tests were passed as shown below

```

1 import unittest
2 from file_to_data import FileToData
3 from io import StringIO
4 import sys
5
6
7 class MyTestCase(unittest.TestCase):
8     def test_show_ast_nodes(self):
9         captured_output = StringIO() # Create StringIO object
10        sys.stdout = captured_output # and redirect stdout.
11        file_to_data = FileToData()
12        file_to_data.read_file("test.py")
13        file_to_data.show_ast_nodes()
14        sys.stdout = sys.__stdout__
15        actual_output = captured_output.getvalue()
16        expected_output = "Module(body=[ClassDef(name='Car', bases=[], keywords=[], body=[FunctionDef(" \
17        "name='__init__', args=arguments(args=[arg(arg='self', annotation=None), " \

```

Terminal: Local

```

C:\Users\harry\Documents\BCDE321Ass2>python test_file_to_data.py
..
-----
Ran 2 tests in 0.002s
OK
C:\Users\harry\Documents\BCDE321Ass2>

```

13.3. Complete and well implemented

- Less than 10 different unit tests

14. Pretty print, i.e., displaying data in chart / diagram, e.g., bar chart, pie chart, UML diagram, etc.

14.1. Used by peers

- File: my_cli.py
 - The def do_pyr_class_diagram(self, file_names): function can display a class diagram based on a python file as shown below:

```

171 # Harry's work
172 def do_pyr_class_diagram(self, file_names):
173     """Generate and display a class diagram in png format from given [png_file_name]
174     self.file_names = file_names
175     python_file_name = file_names[(file_names.find(" ") + 1):]
176     png_file_name = 'classes_' + file_names[0:(file_names.find(" "))] + '.png'
177     try:
178         if path.exists(python_file_name):
179             pyreverse_command = 'pyreverse -ASmn -o png -p ' + file_names
180             subprocess.call(pyreverse_command)
181             print(file_names + ' are done')
182
183         if path.exists(png_file_name):
184             # show png image
185             img = mpimg.imread(png_file_name)
186             fig = plt.imshow(img)
187             fig.axes.get_xaxis().set_visible(False)
188             fig.axes.get_yaxis().set_visible(False)
189             plt.show()
190         else:
191             print("The image of class diagram cannot be generate.")

```

MyCli > do_pyr_class_diagram() > try > if path.exists(python_file_name) > if path.exists(png_file_name)

Project

BCDE321Ass2

John files

CommandLine.py

DiagramCreator.py

my_cli.py

PickleMaker.py

SQLDatabase.py

TestClass.py

Matts files

class.png

class_dia_cli.py

classes.dot

classes.Matt.png

classes_test.png

classes_testoutput.png

DiagramCreator.py

exempldot.dot

exempldot2.dot

file_to_data.py

my_cli.py

file_to_data.py

class_dia_cli.py

img

Harry's work

def do_pyr_class_diagram(self, file_names):

"""Generate and display a class diagram in png format from given [png_file_name]

self.file_names = file_names

python_file_name = file_names[(file_names.find(" ") + 1):]

png_file_name = 'classes_' + file_names[0:(file_names.find(" "))] + '.png'

try:

if path.exists(python_file_name):

pyreverse_command = 'pyreverse -ASmn -o png -p ' + file_names

subprocess.call(pyreverse_command)

print(file_names + ' are done')

if path.exists(png_file_name):

show png image

img = mpimg.imread(png_file_name)

fig = plt.imshow(img)

fig.axes.get_xaxis().set_visible(False)

fig.axes.get_yaxis().set_visible(False)

plt.show()

else:

print("The image of class diagram cannot be generate.")

Figure 1

door

wheel

is_sold()

Door

number

Taxi

color : str

Wheel

number

Terminal: Local

C:\Users\harry\Documents\BCDE321Ass2>python class_dia_cli.py

>>>> pyr_class_diagram testoutput test.py

parsing test.py...

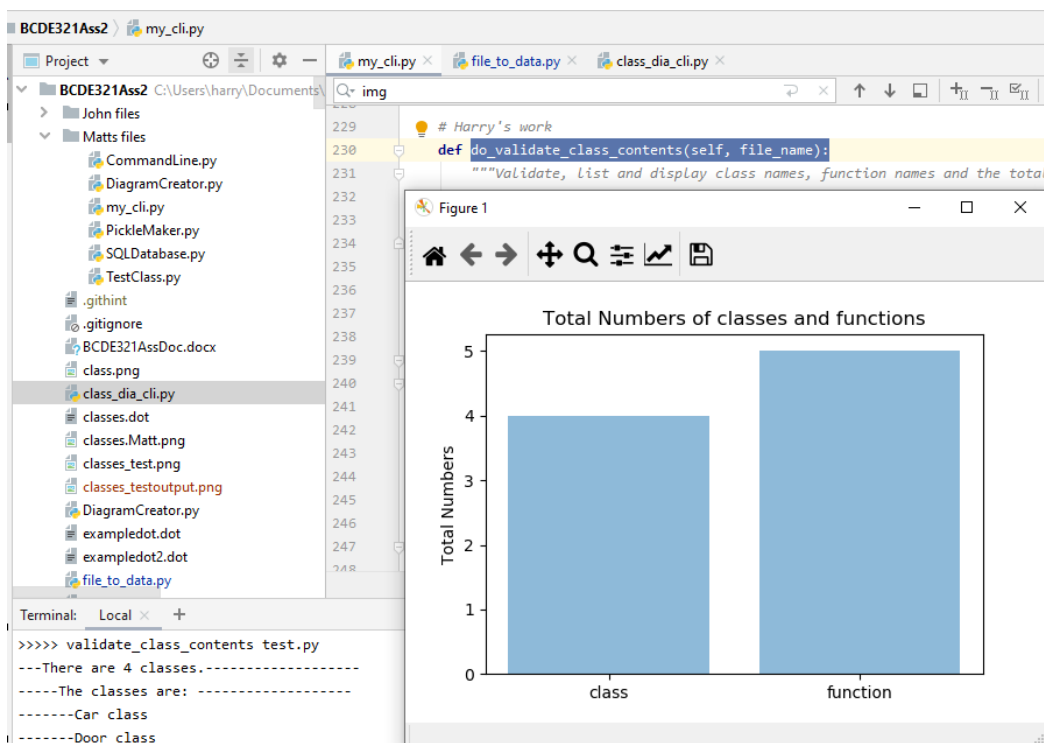
testoutput test.py are done

- The def do_validate_class_contents(self, file_name): function can display a bar diagram to show the total numbers of classes and functions in the input python file as shown below:

```

my_cli.py x file_to_data.py x class_dia_cli.py x
img
229 # Harry's work
230 def do_validate_class_contents(self, file_name):
231     """Validate, List and display class names, function names and the total numbers of them
232     in the given python file. Class and function names are displayed in command line.
233     Total numbers of classes and functions are displayed in a bar graph.
234     Syntax: validate_class_contents [input source code file name.py]"""
235
236     # sample: validate_class_contents test.py
237     num_of_classes = 0
238     num_of_functions = 0
239     try:
240         if path.exists(file_name):
241             self.file_to_data.read_file(file_name)
242             num_of_classes = len(self.file_to_data.tree.body)
243             print("---There are " + str(num_of_classes) + " classes.-----")
244             print("-----The classes are: -----")
245             for my_class in self.file_to_data.tree.body:
246                 print("-----" + my_class.name + " class")
247             for my_class in self.file_to_data.tree.body:
248                 print("-----" + my_class.name + " class has " + str(len(my_class.body)) + " lines")

```



- The def do_dot_2_png(self, input_dot_file_name): function can display a class diagram based on a dot file as shown below:


```

my_cli.py x file_to_data.py x class_dia_cli.py x
Q: img
295 # Harry's work
296 def do_dot_2_png(self, input_dot_file_name):
297     """Generate and display png file from the given dot file.
298     Syntax: dot_2_png [input dot file name.dot]"""
299     try:
300         if path.exists(input_dot_file_name):
301             dot_command = 'dot -Tpng ' + input_dot_file_name + ' -o ' + input_dot_file_name + '.png'
302             subprocess.call(dot_command)
303             print(input_dot_file_name + '.png ' + ' are done')
304             png_file_name = input_dot_file_name + ".png"
305             if path.exists(png_file_name):
306                 # show png image
307                 img = mpimg.imread(png_file_name)
308                 fig = plt.imshow(img)
309                 fig.axes.get_xaxis().set_visible(False)
310                 fig.axes.get_yaxis().set_visible(False)
311                 plt.show()
312             else:
313                 print("The image of class diagram cannot be generate.")
314                 print("Please check with your system administrators.")
MyCli > do_dot_2_png()

```

BCDE321Ass2 > my_cli.py

Project > my_cli.py x file_to_data.py x class_dia_cli.py x

Q: img

```

295 # Harry's work
296 def do_dot_2_png(self, input_dot_file_name):
297     """Generate and display png file from the given dot file.
298     Syntax: dot_2_png [input dot file name.dot]"""
299     try:
300         if path.exists(input_dot_file_name):
301             dot_command = 'dot -Tpng ' + input_dot_file_name + ' -o ' + input_dot_file_name + '.png'
302             subprocess.call(dot_command)
303             print(input_dot_file_name + '.png ' + ' are done')
304             png_file_name = input_dot_file_name + ".png"
305             if path.exists(png_file_name):
306                 # show png image
307                 img = mpimg.imread(png_file_name)
308                 fig = plt.imshow(img)
309                 fig.axes.get_xaxis().set_visible(False)
310                 fig.axes.get_yaxis().set_visible(False)
311                 plt.show()
312             else:
313                 print("The image of class diagram cannot be generate.")
314                 print("Please check with your system administrators.")
MyCli > do_dot_2_png()

```

Figure 1

```

classDiagram
    class Dog {
        +bark() void
    }
    class Cat {
        +meow() void
    }
    class Animal {
        +name string
        +age int
        +die() void
    }
    Dog --|> Animal
    Cat --|> Animal

```

Terminal: Local x +

Please try again! The exception is: 'MyCli' object has no att

C:\Users\harry\Documents\BCDE321Ass2>python class_dia_cli.py

```

>>>> dot_2_png exampledot2.dot
exampledot2.dot.png are done

```

•

14.2. Robustness

- The aforementioned functions have exception handling which checks if the file exists or not and if there is error or not. My program will tell the users if file does not exist in current directory or there are errors as shown in the codes at item 14.1 above.

14.3. Complete and well implemented

- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.
 - There are either one or two blank lines between code blocks according to PEP8.

15. Can save and read data from a database ,e.g., from a database, e.g., SQLite, MySQL and MongoDB.

15.1. Used by peers

- File: my_cli.py
 - The def do_save_py_class_name_and_num_of_functions_to_sqlit(self, file_name): function can save all class name and its number of function in a sqlit database file as shown below. It creates an object of MySQLit class in my_sqlit file to handle the database matters as shown below.

```

my_cli.py x file_to_data.py x PickleMaker.py x class_dia_cli.py x my_sqlit.py x .github x
406
407 # Harry's work
408 def do_save_py_class_name_and_num_of_functions_to_sqlit(self, file_name):
409     """Save all class names and its number of functions to a sqlite database.
410     The classes are extracted from the given python file.
411     Using of my_sqlit database_data command can list out all the data in the database.
412     Syntax: save_py_class_name_and_num_of_functions_to_sqlit [input source code file name.py]"""
413
414     # sample: save_py_class_name_and_num_of_functions_to_sqlit test.py
415
416     num_of_classes = 0
417     num_of_functions = 0
418     try:
419         if path.exists(file_name):
420             my_sqlit = MySqlit('my_sqlite.db')
421             my_sqlit.drop_my_table()
422             my_sqlit.create_my_table()
423
424             self.file_to_data.read_file(file_name)
425             num_of_classes = len(self.file_to_data.tree.body)
426
427             for my_class in self.file_to_data.tree.body:
428                 my_sqlit.my_insert(my_class.name, len(my_class.body))
429
430             my_sqlit.commit_connection()
431             my_sqlit.close_connection()
432
MyCli > do_save_py_class_name_and_num_o...

```

```

my_sqlit.py x my_cli.py x file_to_data.py x PickleMaker.py x class_dia_cli.py x .github x
1 import sqlite3
2
3 class MySqlit:
4
5     def __init__(self, file_name):
6         self.file_name = file_name
7         try:
8             self.conn = sqlite3.connect(self.file_name) #e.g. file_name = 'my_sqlite.db'
9         except Exception as err:
10            print("Please try again! The exception is: ", err)
11        else:
12            print("Database at my_sqlite.db is connected")
13
14    def close_connection(self):
15        self.conn.close()
16        print('mytable is closed')
17
18    def commit_connection(self):
19        self.conn.commit()
20        print('mytable is committed')
21
22    def create_my_table(self):
23        my_cursor = self.conn.cursor()
24        my_cursor.execute("""CREATE TABLE IF NOT EXISTS mytable (
25                            classname text,
26                            numoffunction integer
27                        )""")
28
MySqlit

```

- The def do_my_sqlit_database_data(self,arg):function as shown below can read all class name and its number of function in a sqlite database file which has been saved by using the aforementioned function def do_save_py_class_name_and_num_of_functions_to_sqlit(self, file_name). It creates an object of MySqlit class in my_sqlit file to handle the database matters as shown in the last screenshot above.

```
my_cli.py x my_sqlit.py x file_to_data.py x PickleMaker.py x class_dia_cli.py x .github x
448 # Harry's work
449 def do_my_sqlit_database_data(self, arg):
450     """List all the data stored in the sqlit database by using
451     the save_py_class_name_and_num_of_functions_to_sqlit command.
452     This gives all the pairs of class name and its number of functions.
453     Syntax: my_sqlit_database_data"""
454
455     # sample: save_py_class_name_and_num_of_functions_to_sqlit test.py
456     file_name = 'my_sqlite.db'
457     try:
458         if path.exists(file_name):
459             my_sqlit = MySQLit(file_name)
460             my_sqlit.fetch_all_my_table()
461             my_sqlit.commit_connection()
462             my_sqlit.close_connection()
463         else:
464             print("The database file does not exist in the current directory")
465             print("Please use save_py_class_name_and_num_of_functions_to_sqlit command to create the database")
466             print("Please try again after the database is created!")
467     except Exception as err:
468         print("Please try again! The exception is: ", err)
469
470 # Harry's work
471 def help_my_sqlit_database_data(self):
472     print("\n".join(['List all the data stored in the sqlit database by using ',
473                     'the save_py_class_name_and_num_of_functions_to_sqlit command.',
474                     'This gives all the pairs of class name and its number of functions.',
```

15.2. Robustness

- The aforementioned functions have exception handling which checks if the file exists or not and if there is error or not. My program will tell the users if file does not exist in current directory or there are errors as shown in the codes at item 15.1 above.

15.3. Complete and well implemented

- My code is pythonic. It complies with PEP 8 and is beautiful better than ugly. For example:
 - The code meets the naming convention of PEP 8.
 - There are either one or two blank lines between code blocks according to PEP8.

8 Location of GitHub repository

<https://github.com/harrykhlo/BCDE321Ass2>