# PROG 1700 PYTHON – Final Project

# graphical user interface (GUI) programming

Assignment Value: *15*% of overall course mark.

Due Date: **See due dates designated on the Final Project dropbox on BrightSpace.**

Late submissions will receive the standard late submission penalty as stated in the course outline.

(5% overall deduction per day late, and 0% after assignment handed back to the class.)

### Project – Phased Submission

In no way, shape, or form should this project be attempted in its entirety the day before it is due. If attempted, this will lead to the Earth spinning off of its axis, the complete breakdown of society, and the abolishment of many popular flavours of ice cream, including moon mist. Seriously, this project is not designed to be done in one sitting. The following phases of delivery will be followed, each with their own due dates in subsequent classes:

**Phase 1**: Complete design in QTDesigner of the main form window. Form layout and control naming will be checked. Form design .ui files from Phase 1 successfully converted to .py files and imported into a Visual Studio Code project, and .pyw template implemented to the point that the main windows form launches. Demonstrate pseudocode for handling data flow between text file and application to support phase 3 functionality.

**Phase 2**: The major functionality of the Countries of the World main window completed, including: the loading of the list of countries, and basic country information is displayed when a country is selected, including the country’s name, flag image, population, area (in miles only), population density (in miles only), and world population percentage.

**Phase 3**: All remaining functionality, including mile/KM unit switching for area and population density, country population updating, saving all data changes back to the original text file, and final evaluation of the overall completeness of the program (data validation, formatting, error handling, comments, etc.).

### Project Instructions

#### Use PyQT and Visual Studio Code to create a graphical application (.pyw and supporting .py files) in which you’ll code the solution to this program.

### Submissions

Once you’ve completed the requirements for each phase of the project, commit the .pyw, .py and ui files to your GitHub repository and upload a CommitID text file to Brightspace. Note that you do not need to include the original Countries.txt file and flag image files. For the final phase submission, please **ALSO** add the projects files to a single .ZIP file and upload it to BrightSpace. The .ZIP file name should include your name, W#, section # and Final Project. (ex. DoeJohn\_w0123456\_702\_FinalProject.zip).

**Whether complete or not, you must submit to BrightSpace to receive a mark.**

### Evaluation

To insure the greatest chance of success on this assignment, be sure to check the marking rubrics at the end of this document or in BrightSpace. The rubrics contain the criteria your instructor will be assessing when marking your assignment.

# Program – Countries of the World

Required Resources– Provided by the instructor

* **Resources\_CountriesOfTheWorld.zip**  – A zip file containing the following:
  + **countries.txt** – A text file containing a list of world countries, including name, population and area data.
  + **flags (folder)** – A folder named flags, which contains image files for every country’s flag.

### Program Specifications

Create a graphical program that displays information about any country the user selects from a list.

For any country selected, the application will display the following data:

* Name of the country
* An image of the country’s flag
* The population of the country
* The country’s total area, shown in sq. miles (by default) or sq. kilometres, according to user preference
* The country’s population density per square mile (by default) or square KM, according to user preference
* The country’s percentage of the total world population

The country data being displayed will change automatically whenever a new country is selected from the list.

### Program Functionality

When the application starts, nothing is visible to the user except the File menu. Selecting the Load Countries option from the menu will load all country data from the countries text file into a two-dimensional list. To start, the user will be shown only a list of the country names. Once the user selects any country from the list, a display area will appear that will show information specific to the selected country, as detailed above.

In addition to displaying the country’s flag and basic demographics, the application should include a dropdown list to allow users to change the Total Area display from its default setting of square miles to square kilometres. When this occurs, the country’s Area data should be updated to show the user’s preference of unit.

For the population density section, the density value should display in sq. miles by default. If the user selects either of the Per Square Mile/KM choices, the Population Density data should be updated to show the user’s preference of unit.

Users should be allowed to update the population of the currently selected country by entering a new population value and clicking the Update Population button. If the new data is invalid, an “invalid data” message will be displayed, and the value in the population box will revert back to its original value (from before the attempted changes). If the data is valid, this update the country’s data in the two-dimensional list and display a “Successfully saved to memory” message, but will NOT save the data changes back into the original countries.txt file at that time.

Any population data changes made by the user should only be saved to file when the user either clicks “Save To File” in the menu, or if the program is closed without having already saved the changes. In this case, the program should offer the user the option to save or exit without saving, and do as the user chooses.

## Program Structure

The application should be structured to organize the code into appropriate units of related logic, with an effort to avoid duplication of code and other programming inefficiencies.

To help code’s organizational structure, the following functions/methods are required in your solution:

def LoadCountriesFromFile(dataFileName)

def LoadCountriesListBox()

def SaveCountriesToFile(dataFileName)

def DisplayCountryData(selectedCountryIndex)

def CalculateTotalWorldPopulation()

**Please note:** Your final solution will absolutely need more functions than just those listed above, these requirements are intended as a starting point only.

## Program Quality

Your solution should meet standards most modern users expect from a quality application.

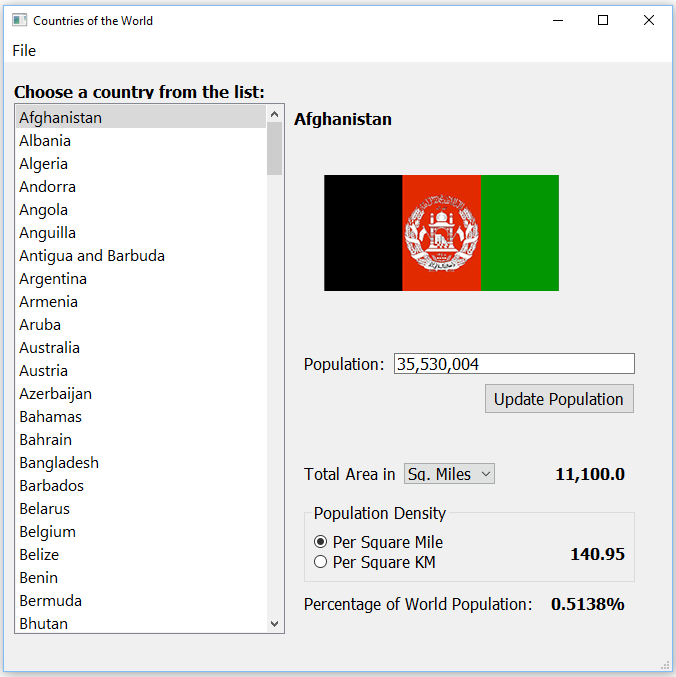
Efforts should be made to:

* Present a clean, intuitive and user-friendly form, with descriptive labels and acceptable layout.
* Include data validation wherever appropriate.
* Include proper formatting of any data/information displayed to the user.
* Use proper scope for all variables.
* Handle for any reasonably-predictable errors that could occur in the application.
* Display helpful and descriptive error messages in a message box, if an exception is encountered.

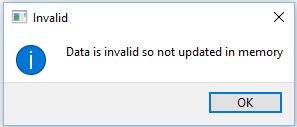
Sample Screenshots

|  |  |
| --- | --- |
| At startup, with country details area hidden | File menu display at startup |
|  |  |
|  |  |

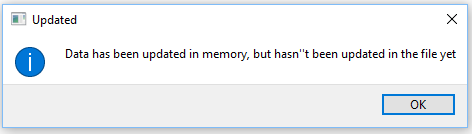
#### After a country has been selected from the countries list



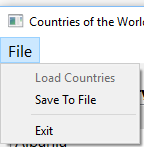
#### Message Popup if updated population data is invalid



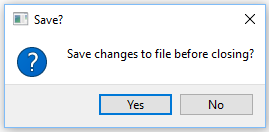
#### Message Popup if updated population data is valid



#### File menu after a country’s population has successfully updated



#### Message Popup if the program is closed before saving changes



### Project Submissions – Phased Marking Explanation

#### Submission Schedule

Each phase of this project (except for the last) will be marked in person, in class with the instructor, during the last classes of the semester. As each phase comes due, your solution to that point will be marked, according to the phased schedule/items listed in the project document.

#### Improving the Previous Phase’s Mark

During each marking phase, you may have the opportunity to improve the marks you received on items from the prior phase. If you passed (60%) the previous phase, you can “make up” points you missed on the previous phase by showing the fixes/improvements you made to correct those errors or omissions. Tracking any prior-phase changes and asking for them to be re-marked is **entirely your own responsibility**. This mark improvement scheme will ONLY apply to items from the previous phase. In other words, if you fix a phase 1 bug, the only time you can get it re-marked is during marking for phase 2. Phase 3 is too late.

#### Cumulative Late Penalties

The phased delivery schedule for this project has been implemented to help students plan and manage their time working on the final project. As such, late penalties will be assessed for each marking phase, and will applied cumulatively to the final project mark. (Ex. A student who was 2 days late for Phase 1, 1 day late for Phase 2, and 1 day late for Phase 3, would incur an overall late penalty of 20% on the final project mark.)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Final Project Rubric - Countries of the World** | | |  |  |  |  |
| **Phase** | **Criteria** | **Unsatisfactory (0 pts)** | **Partially Correct (1 pt)** | **Excellent (2 pts)** | **Marks** | **X** |
| **1** | **Interface Layout** | The GUI layout was not laid out as required.  Too many errors in the layout exist | Most of the GUI matched the requirements  A few small errors in settings exist or controls were not named appropriately | All GUI controls and properties were correctly positioned, sized and set according to specs. Display area hidden at program startup.  All added controls were named appropriately and default control names were not used |  | 2 |
| **1** | **PyQT Import** | Not implemented or contains too many errors. | Implemented, but contains at least one error, or accomplished using inappropriate methods. | Forms/dialogs successfully exported from PyQt .ui files into Python .py library files.  All can be launched from the main window. |  |  |
| **1** | **Pseudocode – Data Handling** | Little to no effort was made to plan out the program using pseudocode | A reasonable but incomplete effort was made to plan out the program using pseudocode | A comprehensive effort was made to plan out the program using pseudocode |  |  |
| **2** | **Text File Loading** | Country data not loaded or contains many errors. | Country data is only partially loaded into the application, not loaded in the required manner, or contains some errors. | All country data is correctly loaded from the text file into a two-dimensional list when the application starts. |  |  |
| **2** | **In-Memory Data Handling** | Country data not loaded into memory, data changes are saved directly to file, or process contains many errors. | Country data is only partially loaded into memory, data is not loaded to memory in the required manner, or process contains some errors. | While app in use, country data is manipulated only in memory, and all data changes are updated only in memory, until an explicit file save is done by user or program exits. |  | 2 |
| **2** | **Country Data – Basic Display** | More than one element of country data missing or incorrect when displayed. | At least one element of country data missing or incorrect when displayed. | Country name, population, total area (in miles) and pop. density (in miles) are all correctly displayed on the form when expected. |  | 3 |
| **2** | **Country Data – Flags** | Flag images not used/displayed, or contain multiple errors. | Flag images are displayed, but with some errors. | Flag images properly displayed when expected, with no errors. |  |  |
| **2** | **World Population Percentage** | World population functionality not implemented, or contains many errors. | World population calculations and display are only partially implemented, do not update as expected, use a hard-coded wordl population, or contain some errors. | Total world population is dynamically calculated from the list of country data and currently selected country’s percentage is correctly calculated and displayed when expected, with no errors. |  | 2 |
| **3** | **Area & Pop. Density – Unit Switching (Mi. to KM & vice versa)** | Total area and pop. density unit switching functionality not implemented, or contains many errors. | Total area and pop. density can be displayed in either unit of measure, but related form controls do not update as expected, or contain small errors in calculations | Total area and pop. density can be displayed in either unit of measure, as per user preference. All related form controls update as expected, with no errors in calculations |  | 2 |
| **3** | **Country Population Updating** | Population value updating is not implemented, or contains multiple errors. | Population value is partially implemented, related form controls do not update as expected, or contains some errors. | Population value is updateable as expected, with no errors. All related form controls update as expected. |  | 2 |
| **3** | **Text File Saving** | Data changes not saved to text file or contains many errors. | Data changes are partially or incorrectly saved to file, or process contains some errors. | Data changes can be properly saved from the in-memory list to the text file. |  |  |
| **3** | **Variable Scope** | Proper scope not used in most cases, or at all. | Proper scope has been used in most, but not all cases. | Proper scope has been considered and used where appropriate throughout the application code. |  |  |
| **3** | **Program Structure** | Application is not well structured, or has more than one required function missing or incorrectly used. | Application is reasonably structured, but contains some inconsistencies, or at least one required function is missing or incorrectly used. | Application structure is well thought out and adheres to required standards. All required functions are included and work together as expected. |  | 2 |
| **3** | **Data Validation** | No data validation used, or contains multiple errors. | Data validation not used wherever appropriate. User can cause an error through incorrect data entry. | Data validation used wherever appropriate. User cannot cause errors through incorrect data entry. |  |  |
| **3** | **Data Formatting** | Very little or no formatting used when displaying data. | Most data uses appropriate formatting when displayed, but not all. | All data displayed to user uses appropriate formatting where appropriate. |  |  |
| **3** | **Error Handling** | Minimal or no error handling used in program. | Error handling used in most cases, but not in all, or user can crash the program, or useful error messaging not communicated to the user. | All reasonably-predictable errors are handled by the application. No program errors encountered during typical use and user informed of any problems via descriptive and helpful message box messages. |  |  |
| **3** | **Comments & Best Coding Practices**  (At least 60% of the functional requirements must be complete) | Little to no organizational or explanatory comments used.  No apparent naming convention was followed or was inconsistently applied. Source code was poorly formatted. | Some organizational or explanatory comments are used, some are meaningful and easily understood. A naming convention was used for part of the program, but deviated often. Improvements could be made. | Organizational or explanatory comments are used extensively, most are meaningful and easily understood. A consistent naming convention was used for most of the program and deviated very little. Source code was clean, consistently well-formatted and easy to read. |  |  |
|  |  |  |  | **Total:** |  | **/50** |