**Module Code:** FC712 **Group:** Comp Group B

**Module Title:** Programming

**Assessment Title:** Programming Project

**Tutor:** Sophie Norman

**GUID:** 2840479b

**Submission Date:** 02.03.2023

**Declaration of Ownership:** I confirm that this assignment is my own work. Where I have referred to academic sources, I have provided in-text citations and included the sources in the final reference list.

**Table of Contents**

**Section 1.1: Task 1 Algorithm 3**

**Section 1.2: Task 1 Technical Overview** **5**

**Section 1.3: Task 1 Testing 5**

**Section 1.4: Task 1 Summary6**

**Section 2.1: Task 2 Algorithm 7**

**Section 2.1.1: Functions7**

**Section 2.1.2: Main Body12**

**Section 2.2: Task 2 Technical Overview16**

**Section 2.3: Task 2 Testing17**

**Section 2.4: Task 2 Summary17**

**Section 3.1: Task 1 Code Listing18**

**Section 3.2: Task 2 Code Listing18**

**Section 4: References18**

**Section 1.1: Task 1 Algorithm**

Algorithms used in this project will be presented in a pseudocode:

Define class ReusableForm(Form):

Name is a string field with a DataRequired validator

p-number is a string field with a DataRequired validator

email is a string field with a DataRequired validator

understanding is a radio field with a DataRequired validator and 5 options from strongly agree to strongly disagree

tutor\_satidfaction is a radio field with a DataRequired validator and 5 options from strongly agree to strongly disagree

pass\_course is a Boolean field with a DataRequired validator

reflection is a textAreaField with a DataRequired validator

Def write\_to\_disk(course, p\_number, name, email, understanding, tutor\_satisfaction, pass\_course, reflection):

Open course + .txt as data

Split reflection (text) by ‘\n’

For each line in reflection:

Remove ‘\r’ from a line

Write to data all function args

Close data

Define a route app.route(‘/user/<name>’)

Def user(name):

Render template user.html with name in html being changed for name in argument

Define a route app.route(‘/user/course/<course\_name>’), define methods POST and GET

Def course(course\_name):

Create reusableForm as form

If method is POST:

Get form data for each field defined in reusableForm class

If form is validated:

Write gotten information to file with write\_to\_disk

Flash thank you message

If not validated:

Flash error message

Render course.html template with course\_name and form in template being changed to course\_name and form from arguments

**Furthermore, in templates, apart from markdown, following algorithm is used:**

Get flashed messages and put them in messages

If there are messages:

For each message:

If ‘error’ is not in message:

Create success message on screen

If ‘error’ is in messages:

Create error message on screen

**Section 1.2: Task 1 Technical Overview**

|  |  |
| --- | --- |
| **Key Variables and Data structures** | |
| Name, p\_number, email, understanding, tutor\_satisfaction, pass\_course, reflection | Fields, including text fields, text area fields and radio fields. Responsible for rendering and data conversion. |
| Form\_course | A reusable form instance |
| {{course\_name}}, {{ form.csrf }} | Used in html templates, substituted with the name of the course and form respectively |
| **Key Functions and Algorithms** | |
| {% with messages = <…> {% endwith %} | This algorithms is used to get flashed messages and display them on the webpage |
| <link href="{{ url\_for('static', filename='css/css.css') }}" rel = "stylesheet" /> | This statement is used to get a link to css file and then link it to the webpage |
| **User(name), course(course\_name)** | **These functions render selected templates, user.html and course.html respectively. Additionally, if request method is POST, course(course\_name) retrieves information from the form and writes it to the save file** |
| **Write\_to\_disk(course, p\_number, name, email, understanding, tutor\_satisfaction, pass\_course, reflection)** | **This function opens a txt file for selected course, separates reflection text and writes all arguments to the file** |

**Section 1.3: Task 1 Testing**

I have approached testing process very diligently. The main objective was to test for errors if any of the fields are not filled in. Validators help with detecting this usually, but during the testing process I have found that radio buttons cannot be validated the way other fields can. I therefore decided to set some buttons to be selected by default, and it solved the error. I have tested the app if certain templates and save files were missing, and solved errors that were unveiled during this testing

Furthermore, I have asked my peers to also test my forms. They were satisfied and couldn’t find any bugs. I therefore concluded that my project has withstood testing necessary to call it a success. The results of testing are in files labeled: engineering.txt, programming.txt and mathematics.txt.

**Section 1.4: Task 1 Summary**

As I mentioned above, I consider my project to be a success. It was based on information provided to us in worksheets, during labs and lectures, as well as documentation for Flask and WTForms. I believe that the solution works great, however the project would benefit from further development and implementation of smaller additional features, for example displaying information from save files or adding more questions to deepen the understanding of how well the student has done on course.

**Section 2.1: Task 2 algorithm**

Some of the most significant algorithms used in this project will be presented in a pseudocode:

**Section 2.1.1: Functions**

Def import\_chars():

TRY

opening store\_char as save\_file

read lines of save\_file to in\_list

EXCEPT if file not found:

Show warning in a messagebox

ELSE

FOR each line/character

Remove ‘\n’ from string

Split string by ‘ – ‘ separator

Write character as char\_dict[name of character] = {class: 1st element, race: 2nd element, weapon: 3rd element, str: 4th element, con: 5th element, dex: 6th element, int: 7th element, wis: 8th element

Close save file

Def update\_layers(images\_present):

If images are present:

Use global char\_race, char\_class, char\_weapon

race\_layer = image created on canvas C, image = char\_race

class\_layer = image created on canvas C, image = char\_class

weapon\_layer = image created on canvas C, image = char\_weapon

Else:

Draw rectangles instead of images

Def spawn\_char(name):

Use global char\_race, char\_class, char\_weapon

Set text in entry\_var to be name

For each stat in stat\_dict

Assign stat in stat\_dict to be value of the same stat in char\_dict[name]

Call assign\_stats with stat\_dict as an argument

Call set\_final\_scores

Call set\_race from current character’s race

Call set\_class from current character’s race

Call set\_weapon from current character’s race

Def set\_race(race\_name):

Use global char\_race, char\_class, char\_weapon

Curr\_dict[‘race’] = race\_name

Set images\_present to True

TRY

Open race\_name.png as char\_race

EXCEPT if unable to open:

Set images\_present to False

FINALLY:

Set race\_info\_var to be race\_dict[race\_name]

For each stat set racial modifier according to race\_bonus dictionary

Update\_layers(images present)

Def set\_class(class\_name):

Use global char\_race, char\_class, char\_weapon

Curr\_dict[‘class’] = class\_name

Set images\_present to True

TRY

Open class\_name.png as char\_class

EXCEPT if unable to open:

Set images\_present to False

FINALLY:

Call change\_info()

Call update\_weapon\_menu()

Update\_layers(images\_present)

Def set\_weapon(weapon\_name):

Use global char\_race, char\_class, char\_weapon

Curr\_dict[‘weapon’] = weapon\_name

Set images\_present to

TRY

Open weapon\_name.png as char\_class

EXCEPT if unable to open:

Set images\_present to False

FINALLY:

If weapon\_name is not blank:

Set weapon\_info\_var to be weapon stats from weapon\_stat

Else:

Set weapon\_info\_var to be ‘’

Update\_layers(images\_present)

Def update\_weapon\_menu(class\_name):

Dict c\_to\_w holds class to weapon correlation

List all\_weapon holds all weapons

Delete current buttons in weapon menu

For each weapon put a radiobutton with function lambda: set\_weapon

For each weapon in all weapons:

If weapon is not in c\_to\_w[class\_name]:

Delete weapon button

Def merge(list1, list2, list3):

i1, i2, i3 = 0, 0, 0 for iteration

n1, n2 = length of list1, length of list2

while i1 less than n1 and i2 less than n2:

if current element of list1 less than current element of list2

current element of list1 is new current element of list3

iterate through list1

else:

current element of list2 is new current element of list3

iterate through list2

iterate through list3

while i1 less than n1:

current element of list1 is new current element of list3

iterate through list1

iterate through list3

while i2 less than n2:

current element of list2 is new current element of list3

iterate through list2

iterate through list3

Def sort\_m(list):

N = length of list

If N greater than 1:

List1 is first half of list

List2 is second half of list

Sort\_m(list1)

Sort\_m(list2)

Merge list1 and list2 into list

Def assign\_stats(dict):

For each stat configure corresponding labels with corresponding value from dict

Def roll\_stats()

Arr is a temporary list

Global dict stat\_dict

FOR 6 times:

Roll is temporary list

FOR 4 times:

Generate random values from 1 to 6 and append to roll

Sort roll with sort\_m

Add to arr sum of all elements of roll apart from zeroth (=lowest)

For each element in arr make it string

FOR each stat in stat\_dict:

Set stat\_dict[current] to be current value from arr

Iterate through arr

Def set\_final\_scores():

IF stat\_dict[STR] is not empty => if stat\_dict is not empty:

For each stat:

set stat\_var to be string of sum of integers from stat\_dict[stat] and stat\_rmod\_var (racial modifier)

For each stat:

Tmp = (stat\_var – 10) / 2 round down

Configure L\_stat\_mod to be tmp with ‘+’ in front, unless there is ‘-‘ in front

IF current class isn’t blank:

Call change\_info()

Def add\_new\_character():

Name = text in entry field

IF character is ready (determined by char\_ready):

By default character doesn’t exist

IF the name is in char\_dict keys:

Character exists

Add character to char\_dict with name as key

For every stat in stat\_dict:

Add stat to char\_dict

Call export\_chars()

If the character didn’t exist:

Add button to frame\_char\_sel which calls spawn\_char

Pack button

configure label l\_error to text from char\_ready

def clear\_error\_message():

wait 3 seconds

configure label l\_error to text ‘’

create thread with clear\_error\_message()

**Section 2.1.2: main body**

TRY:

Open parchment.png as bg

Bg Image is created on canvas

EXCEPT if image not found:

Error is shown in message box

Canvas colour is set to #e3c076

TRY:

Open frame.png as frame\_img

Frame\_image is created on canvas

EXCEPT if image not found:

Error is shown in message box

Substitute frame is created via create\_rectangle

l\_load is a label with instructions for frame\_char\_sel

l\_load is packed()

for each character in char\_dict():

l\_char button created that calls spawn\_char(character)

l\_char is packed

l\_new\_char is a button that calls add\_new\_character

l\_new\_char is packed

l\_error Is a label that displays error messages or success messages

l\_error is packed

l\_name\_text is a label with instructions for frame\_rcw\_sel

l\_name\_text is packed to grid

entry\_var is a StringVar() that holds name of the character

e\_name is entry field with entry\_var as textvariable

e\_name is packed to grid

m\_race is a menu button in frame\_rcw\_sel with text ‘select race:’  
m\_race is packed to grid

m\_race.menu is a menu

raceVar is a StringVar() that holds race selected with radiobuttons

for each race:

a radiobutton is created, race name is the label, command is lambda that calls set\_race(race selected with race\_var)

labels are created with raised relief and packed to grid in a line:

‘Ability name’

‘Generated score’

‘Racial modifier’

‘Ability score’

‘Ability Modifier’

For each ability:

L\_ability is a label with ability name and raised relief

L\_ability is packed into grid

L\_ability\_score is a label with ability score or ‘ ’ default

L\_ability\_score is packed to grid

L\_ability\_race is a label with racial modifier for ability

Ability\_rmod\_var is set to default “+0”

L\_ability\_race is packed to grid

L\_ability\_f\_score is a label with final scores

L\_ability\_f\_score is packed to grid

L\_ability\_mod is a label with ability modifier

L\_ability\_mod is packed to grid

B\_roll is a button in frame\_stat that calls roll\_stats

B\_roll is packed to grid

FOR each row from 0 to 6

Row padding is set to 30

**Section 2.2: Task 2 Technical Overview**

In this and further sections, “race, class, weapon” are referred to as “Attributes”, and “STR, CON, DEX, INT, WIS, CHA” are referred to as “Stats”

|  |  |
| --- | --- |
| **Key Variables and Data Structures** | |
| global char\_race, char\_class, char\_weapon | Because images are created in functions, once function is finished, garbage collector empties variables created in functions. For images to stay on screen, these global variables are needed |
| char\_dict | dictionary of dictionaries, in which characters are stored after being imported from save file |
| curr\_dict, stat\_dict | dictionaries, with current attributes and stats |
| Weapon\_stat | dictionary with information about weapons |
| Str\_var, con\_var, dex\_var, int\_var, wis\_var, cha\_var | StringVars, hold final scores of their respective Stats, currently on screen |
| **Key Functions** | |
| import\_chars() | transfers characters from save file to char\_dict |
| update\_layers() | uses global variables and redraws character in correct layer order |
| spawn\_char(name) | this function sets selected character to be the current one displayed |
| set\_race(race\_name)/class/weapon | this function updates current character's attribute |
| sort\_m(lst\_inp), merge(lst1, lst2, lst3) | a merge sort algorithm that recursively calls itself to sort a list |
| roll\_stats() | this function uses classic 'roll 4d6 drop lowest' algorithm to generate stats for the character |
| set\_final\_scores() | final scores are calculated and displayed, ability modifiers are calculated and displayed |
| add\_new\_char() | this function saves current character to char\_dict |
| char\_ready() | checks if the character has his class, race, weapon, name and stats selected, returns tuple with true or false and error or success message |

**Section 2.3: Task 2 Testing**

I tried creating different characters, removing crucial files, which led to errors and flaws being discovered. Currently all errors are being handled with exception handling or if statements. All other ways of interacting with program have been thoroughly investigated, and character creation process goes uninterrupted in most cases.

I have asked people around me, including those who know little about programming, to test it. They noted the clarity of the interface, and since my program is targeted at casual users using it, I believe it to be a success. The results of testing and character creation are in save\_file.txt

**Section 2.4: Task 2 Summary**

The initial project idea was bigger in scale, but during the development process it became evident that it would be impossible to be able to finish it in time and with due diligence, so the second part was removed, and only the character creator was left. Certain aspects of the code had to be worked on more than anticipated, for example creating buttons to load characters.

Overall, I am happy with the project. Nevertheless, I know that certain data structures could be improved, and the process of creating/assigning stats could also probably be optimized, and the project would benefit from using classes and objects.

**Section 3.1: Task 1 Code Listing**

1. Materials provided on the VLE
2. <https://wtforms.readthedocs.io/en/2.3.x/>

**Section 3.2: Task 2 Code Listing**

1. Materials provided on VLE
2. <https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html>
3. <https://stackoverflow.com/questions/10865116/tkinter-creating-buttons-in-for-loop-passing-command-arguments>
4. <https://stackoverflow.com/questions/55551007/how-to-make-a-timer-in-python-without-freezing-up-the-entire-code>

**Section 4: Reference**

1. **Heinsoo, R., Collins, A. and Wyatt, J. (2008). Dungeons & dragons player’s handbook: arcane, divine, and martial heroes: roleplaying game core rules. Renton, Wa: Wizards Of The Coast.**
2. **anzeljg.github.io. (n.d.). Tkinter 8.5 reference: a GUI for Python. [online] Available at: https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html.**
3. **wtforms.readthedocs.io. (n.d.). WTForms — WTForms Documentation (2.3.x). [online] Available at:** [**https://wtforms.readthedocs.io/en/2.3.x/**](https://wtforms.readthedocs.io/en/2.3.x/)**.**
4. **Stack Overflow. (n.d.). How to make a timer in python without freezing up the entire code. [online] Available at: https://stackoverflow.com/questions/55551007/how-to-make-a-timer-in-python-without-freezing-up-the-entire-code.**
5. **Stack Overflow. (n.d.). python - tkinter creating buttons in for loop passing command arguments. [online] Available at: https://stackoverflow.com/questions/10865116/tkinter-creating-buttons-in-for-loop-passing-command-arguments.**