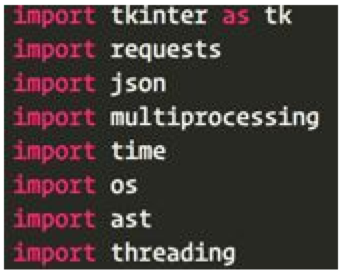
1. Imported modules:



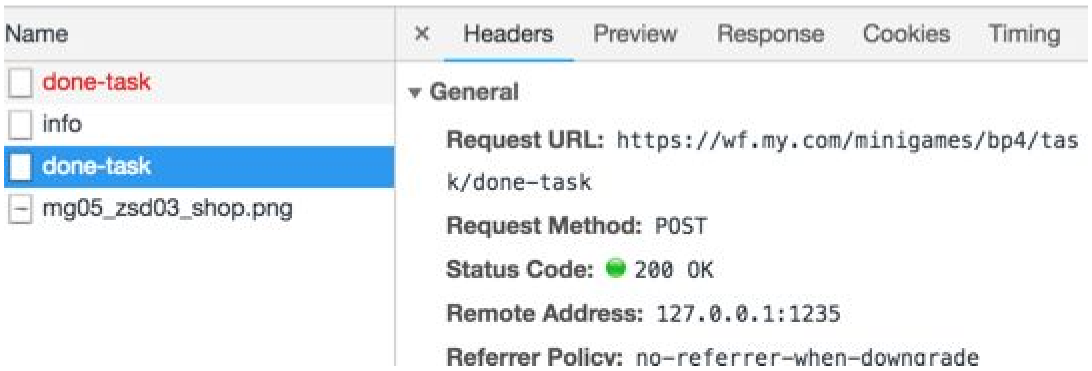
```
import tkinter as tk
import requests
import json
import multiprocessing
import time
import os
import ast
import threading
```

- tkinter is the interface of GUI framework for python. It is used to create all the GUI elements in the program.
- requests is the python module to send http/https requests. It is used to retrieve needed information from game server and send request to perform certain action in this program.
- threading is the python module to create threads that can be ran at the same time with the main process. It is used to handle the loops to avoid the freeze of GUI.
- json module is the json file processing library for python.
- time module is used to provide sleep function in this program.
- os module is used to provide system information in this program.
- ast is used to transfer variable type in this program.
- multiprocessing is used in a case to replace threading in this process as it is more convenient to terminate.

2. Integrated Development Environment (IDE):
- The IDE used during the development is Sublime Text editor as it provides multi-line cursor and mass function name rewriting.

3. Techniques used:
- Dictionary
- List
- while Loop
- for Loop
- global variable
- Inheritance
- requests: GET, POST, session
- threading: Thread
- multiprocessing: Process
- tkinter: TK, frame, label, input, button, canvas, StringVar, IntVar
- file: open, read, seek, write
- os: path

4. Network requests analysis
   - The network requests analysis is done in Chrome with its built-in developer tools
   - done_task:



Name | × Headers | Preview | Response | Cookies | Timing

done-task
info
done-task
mg05_zsd03_shop.png

▾ General

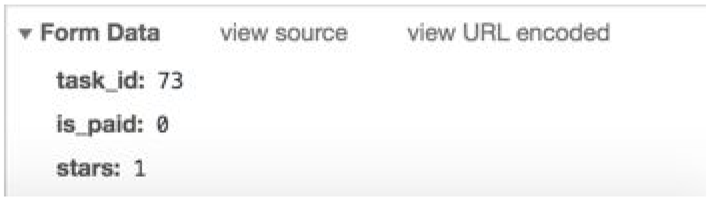
**Request URL:** `https://wf.my.com/minigames/bp4/task/done-task`

**Request Method:** `POST`

**Status Code:** ● `200 OK`

**Remote Address:** `127.0.0.1:1235`

**Referrer Policy:** `no-referrer-when-downgrade`



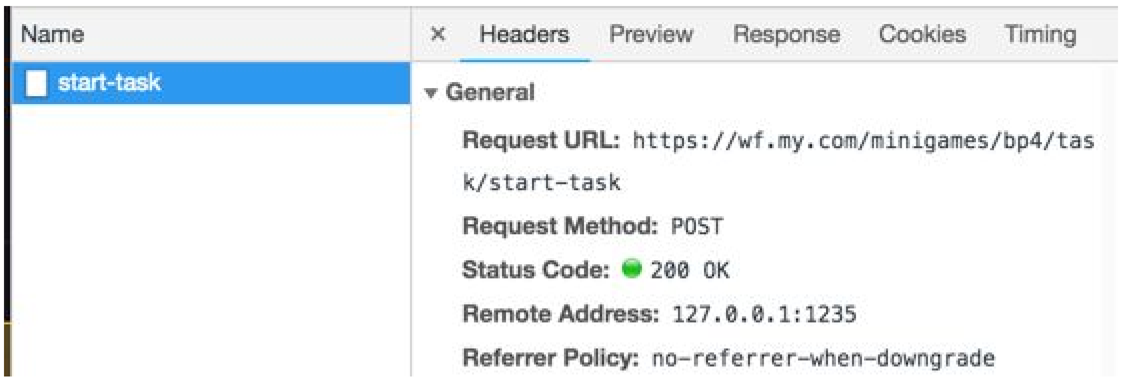
▾ **Form Data** view source view URL encoded

**task_id:** `73`

**is_paid:** `0`

**stars:** `1`

   - start_task:



Name | × Headers | Preview | Response | Cookies | Timing

start-task

▾ General

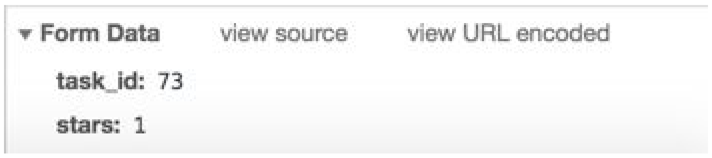
**Request URL:** `https://wf.my.com/minigames/bp4/task/start-task`

**Request Method:** `POST`

**Status Code:** ● `200 OK`

**Remote Address:** `127.0.0.1:1235`

**Referrer Policy:** `no-referrer-when-downgrade`

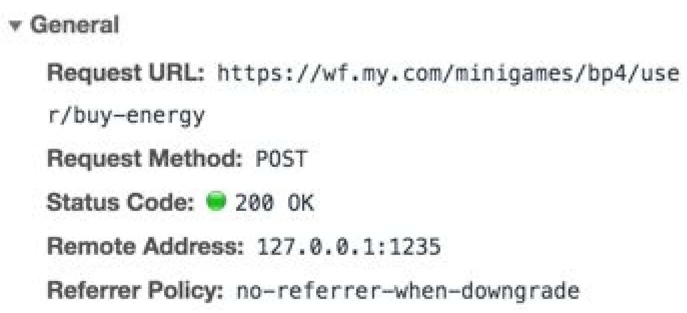


▾ **Form Data** view source view URL encoded

**task_id:** `73`

**stars:** `1`

   - energy_refill:



▾ General

**Request URL:** `https://wf.my.com/minigames/bp4/user/buy-energy`

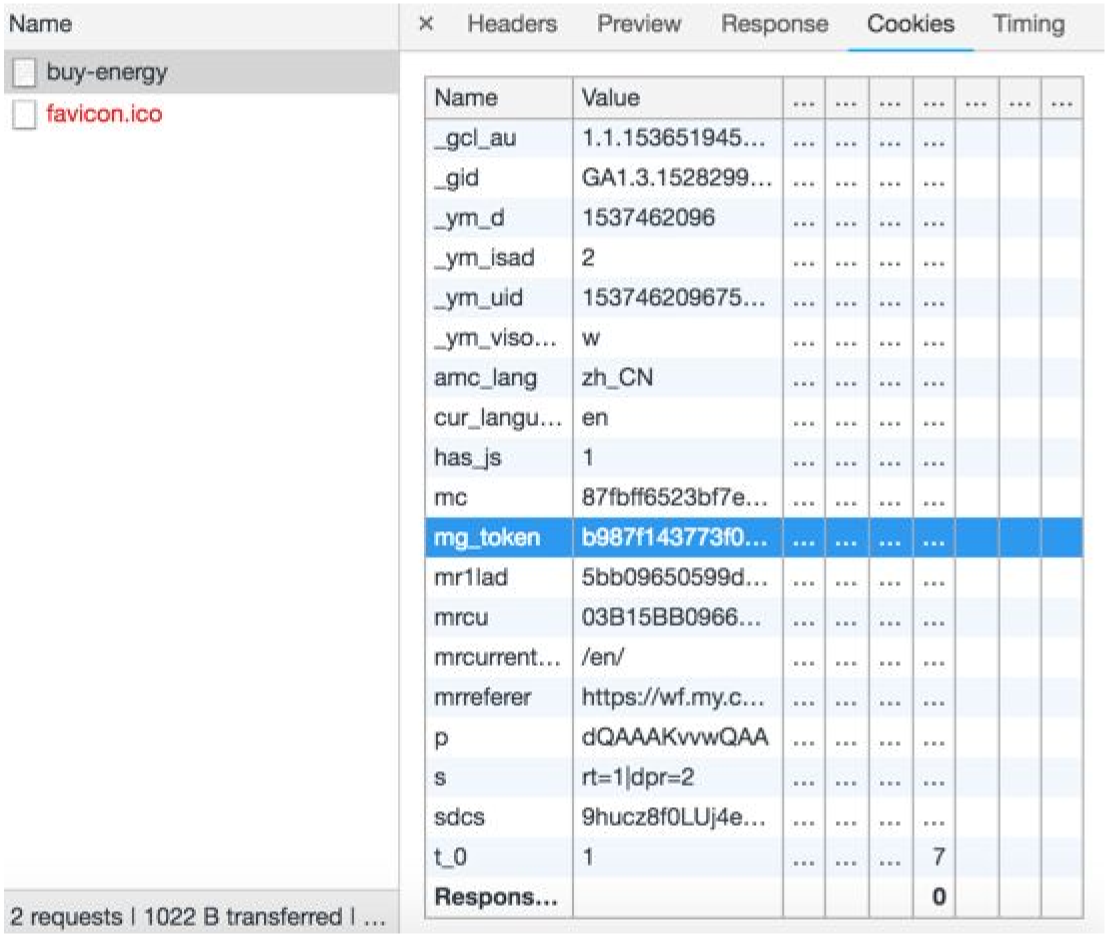
**Request Method:** `POST`

**Status Code:** ● `200 OK`

**Remote Address:** `127.0.0.1:1235`

**Referrer Policy:** `no-referrer-when-downgrade`

   - cookies:

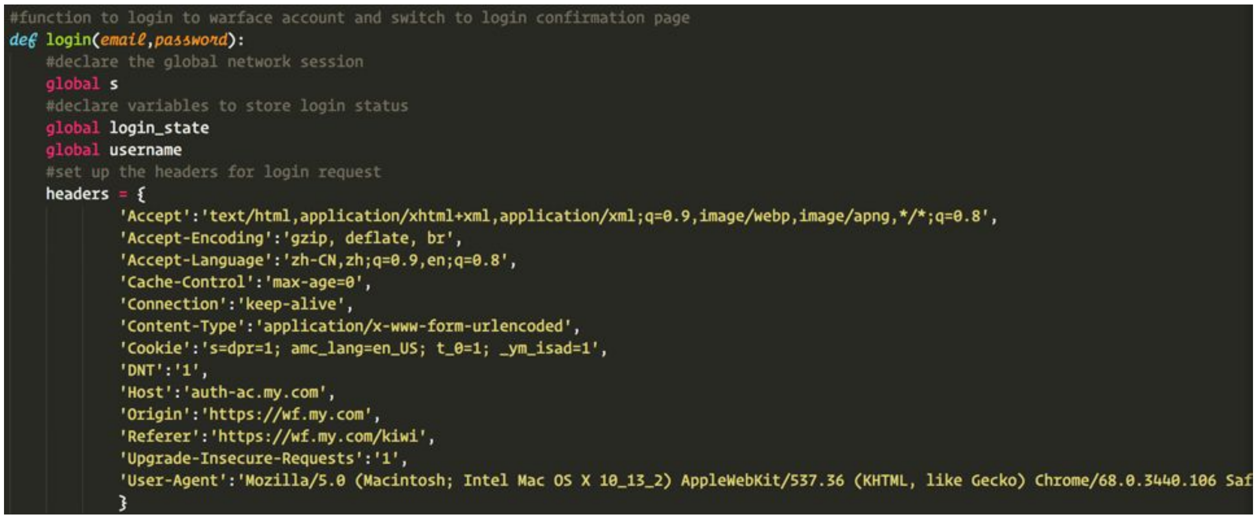


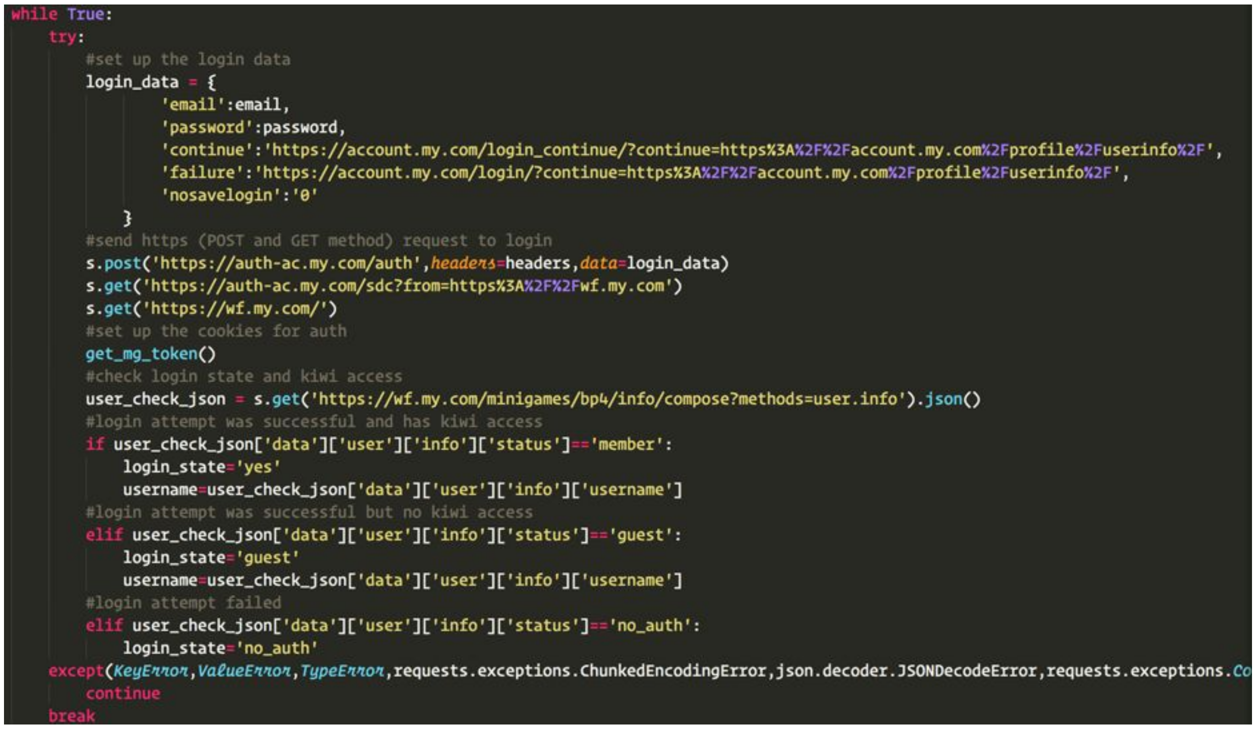
Name
buy-energy
favicon.ico

2 requests | 1022 B transferred | …

| Name | Value | … | … | … | … | … | … | … |
|---|---|---|---|---|---|---|---|---|
| _gcl_au | 1.1.153651945… | … | … | … | … | | | |
| _gid | GA1.3.1528299… | … | … | … | … | | | |
| _ym_d | 1537462096 | … | … | … | … | | | |
| _ym_isad | 2 | … | … | … | … | | | |
| _ym_uid | 153746209675… | … | … | … | … | | | |
| _ym_viso… | w | … | … | … | … | | | |
| amc_lang | zh_CN | … | … | … | … | | | |
| cur_langu… | en | … | … | … | … | | | |
| has_js | 1 | … | … | … | … | | | |
| mc | 87fbff6523bf7e… | … | … | … | … | | | |
| mg_token | b987f143773f0… | … | … | … | … | | | |
| mr1lad | 5bb09650599d… | … | … | … | … | | | |
| mrcu | 03B15BB0966… | … | … | … | … | | | |
| mrcurrent… | /en/ | … | … | … | … | | | |
| mrreferer | https://wf.my.c… | … | … | … | … | | | |
| p | dQAAAKvvwQAA | … | … | … | … | | | |
| s | rt=1\|dpr=2 | … | … | … | … | | | |
| sdcs | 9hucz8f0LUj4e… | … | … | … | … | | | |
| t_0 | 1 | … | … | … | 7 | | | |
| **Respons…** | | | | | **0** | | | |

5. login Function


```
#function to login to warface account and switch to login confirmation page
def login(email,password):
    #declare the global network session
    global s
    #declare variables to store login status
    global login_state
    global username
    #set up the headers for login request
    headers = {
            'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
            'Accept-Encoding':'gzip, deflate, br',
            'Accept-Language':'zh-CN,zh;q=0.9,en;q=0.8',
            'Cache-Control':'max-age=0',
            'Connection':'keep-alive',
            'Content-Type':'application/x-www-form-urlencoded',
            'Cookie':'s=dpr=1; amc_lang=en_US; t_0=1; _ym_isad=1',
            'DNT':'1',
            'Host':'auth-ac.my.com',
            'Origin':'https://wf.my.com',
            'Referer':'https://wf.my.com/kiwi',
            'Upgrade-Insecure-Requests':'1',
            'User-Agent':'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Saf
            }
```



```
while True:
    try:
        #set up the login data
        login_data = {
                'email':email,
                'password':password,
                'continue':'https://account.my.com/login_continue/?continue=https%3A%2F%2Faccount.my.com%2Fprofile%2Fuserinfo%2F',
                'failure':'https://account.my.com/login/?continue=https%3A%2F%2Faccount.my.com%2Fprofile%2Fuserinfo%2F',
                'nosavelogin':'0'
            }
        #send https (POST and GET method) request to login
        s.post('https://auth-ac.my.com/auth',headers=headers,data=login_data)
        s.get('https://auth-ac.my.com/sdc?from=https%3A%2F%2Fwf.my.com')
        s.get('https://wf.my.com/')
        #set up the cookies for auth
        get_mg_token()
        #check login state and kiwi access
        user_check_json = s.get('https://wf.my.com/minigames/bp4/info/compose?methods=user.info').json()
        #login attempt was successful and has kiwi access
        if user_check_json['data']['user']['info']['status']=='member':
            login_state='yes'
            username=user_check_json['data']['user']['info']['username']
        #login attempt was successful but no kiwi access
        elif user_check_json['data']['user']['info']['status']=='guest':
            login_state='guest'
            username=user_check_json['data']['user']['info']['username']
        #login attempt failed
        elif user_check_json['data']['user']['info']['status']=='no_auth':
            login_state='no_auth'
    except(KeyError,ValueError,TypeError,requests.exceptions.ChunkedEncodingError,json.decoder.JSONDecodeError,requests.exceptions.Co
        continue
    break
```

- To avoid being set into infinite loop by the server, headers are set to simulate the environment of Google Chrome on macOS High Sierra.

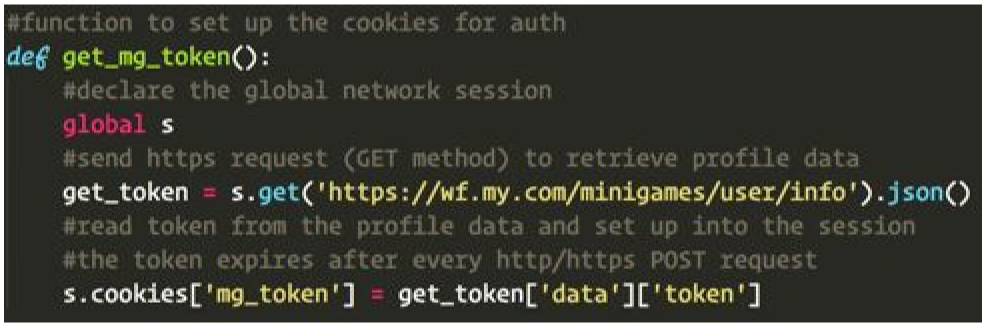
- To avoid failure of login due to frequent attempt, in which condition the server will set the program into infinite loop, or returning a non-standard json file, a while Loop combining with exceptions is used. In the case of a failure, the login process will try again until it succeeds.

- The HTTP POST method is used to send the login information. And HTTP GET method is used to finish downloading data for authorization.

- get_mg_token is a function used to get the token and store in cookies to authorize for event data. This will be explained in below section.

- To check if the login was successful and if the account used logged into has the access to the event, HTTP GET method is used to download user's profile, which will be in json format. The program will check the status info and store the login status into the global variable, login_state.

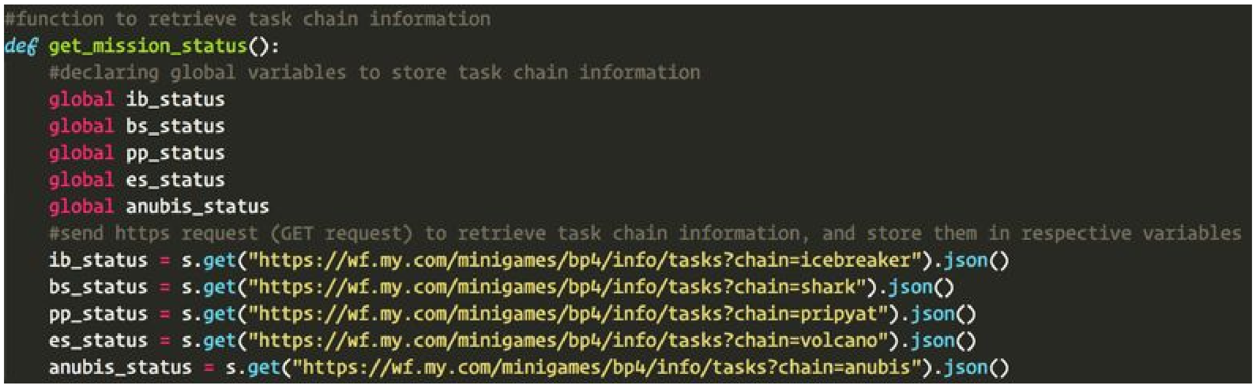
6. get_mg_token function



```python
#function to set up the cookies for auth
def get_mg_token():
    #declare the global network session
    global s
    #send https request (GET method) to retrieve profile data
    get_token = s.get('https://wf.my.com/minigames/user/info').json()
    #read token from the profile data and set up into the session
    #the token expires after every http/https POST request
    s.cookies['mg_token'] = get_token['data']['token']
```

- HTTP GET method is used to fetch user profile
- The profile is returned in json format, the token is set into cookies of the requests session

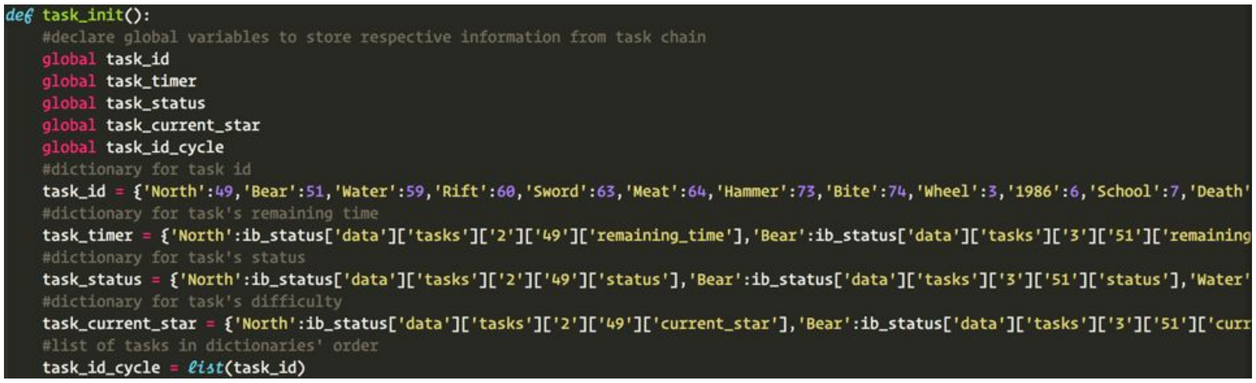
7. get_mission_status function



```python
#function to retrieve task chain information
def get_mission_status():
    #declaring global variables to store task chain information
    global ib_status
    global bs_status
    global pp_status
    global es_status
    global anubis_status
    #send https request (GET request) to retrieve task chain information, and store them in respective variables
    ib_status = s.get("https://wf.my.com/minigames/bp4/info/tasks?chain=icebreaker").json()
    bs_status = s.get("https://wf.my.com/minigames/bp4/info/tasks?chain=shark").json()
    pp_status = s.get("https://wf.my.com/minigames/bp4/info/tasks?chain=pripyat").json()
    es_status = s.get("https://wf.my.com/minigames/bp4/info/tasks?chain=volcano").json()
    anubis_status = s.get("https://wf.my.com/minigames/bp4/info/tasks?chain=anubis").json()
```

- The function is used to fetch the status of the character. The returned json file contains information of whether a task if currently active, and the remaining time until completion.
- HTTP GET method is used to fetch the result.

8. task_init function



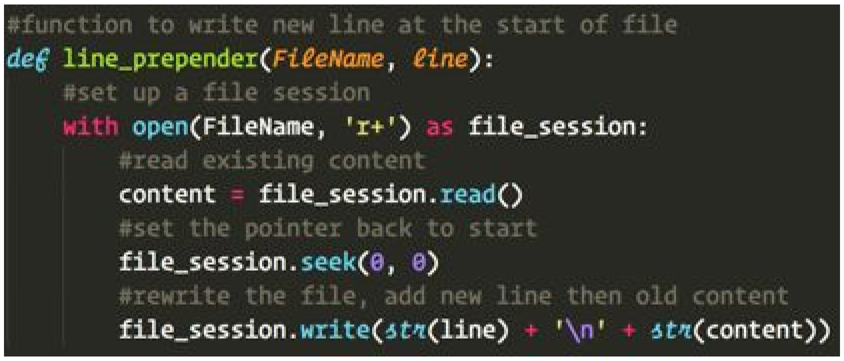
```python
def task_init():
    #declare global variables to store respective information from task chain
    global task_id
    global task_timer
    global task_status
    global task_current_star
    global task_id_cycle
    #dictionary for task id
    task_id = {'North':49,'Bear':51,'Water':59,'Rift':60,'Sword':63,'Meat':64,'Hammer':73,'Bite':74,'Wheel':3,'1986':6,'School':7,'Death'
    #dictionary for task's remaining time
    task_timer = {'North':ib_status['data']['tasks']['2']['49']['remaining_time'],'Bear':ib_status['data']['tasks']['3']['51']['remaining
    #dictionary for task's status
    task_status = {'North':ib_status['data']['tasks']['2']['49']['status'],'Bear':ib_status['data']['tasks']['3']['51']['status'],'Water'
    #dictionary for task's difficulty
    task_current_star = {'North':ib_status['data']['tasks']['2']['49']['current_star'],'Bear':ib_status['data']['tasks']['3']['51']['curr
    #list of tasks in dictionaries' order
    task_id_cycle = list(task_id)
```

(The screenshot does not show the full content of dictionary)

- There's 20 tasks in total, and all their information are fetched by get_mission_status (the function explained above).

- This function uses 4 dictionary to store task's ID, remaining time, status, and its difficulty.
- task_id_cycle is a list of all tasks' name in the same order as dictionaries.

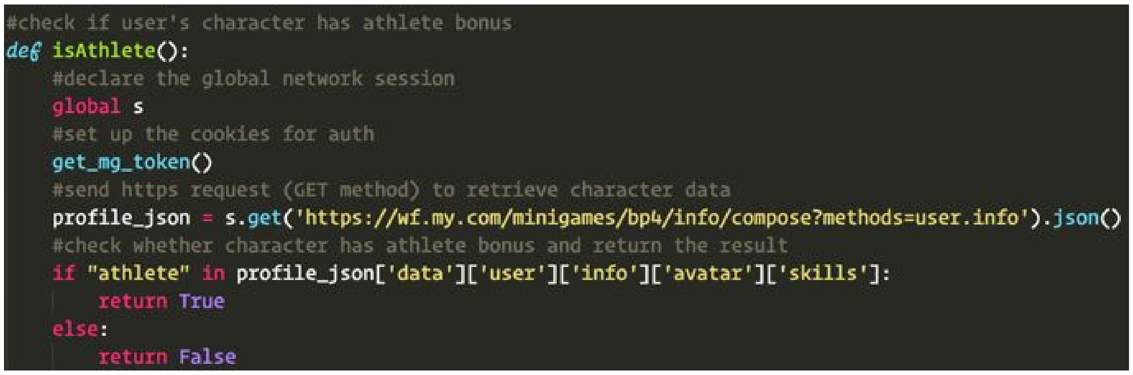
9. line_prepender function



```python
#function to write new line at the start of file
def line_prepender(FileName, line):
    #set up a file session
    with open(FileName, 'r+') as file_session:
        #read existing content
        content = file_session.read()
        #set the pointer back to start
        file_session.seek(0, 0)
        #rewrite the file, add new line then old content
        file_session.write(str(line) + '\n' + str(content))
```

- The function works by storing the old file content, then rewrite it with new content plus the old content.

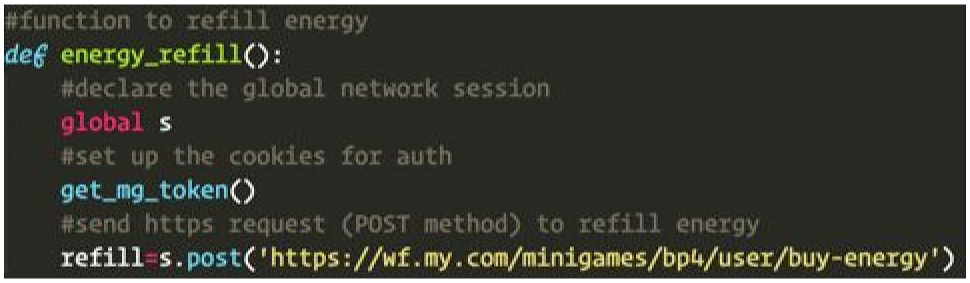
10. isAthlete function



```python
#check if user's character has athlete bonus
def isAthlete():
    #declare the global network session
    global s
    #set up the cookies for auth
    get_mg_token()
    #send https request (GET method) to retrieve character data
    profile_json = s.get('https://wf.my.com/minigames/bp4/info/compose?methods=user.info').json()
    #check whether character has athlete bonus and return the result
    if "athlete" in profile_json['data']['user']['info']['avatar']['skills']:
        return True
    else:
        return False
```

- athlete is a bonus status in the event, when the character has it, the energy needed to activate a task will be reduced.
- To get the character information, the mini game token, mg_token, is needed during the authorization. It will be fetched by get_mg_token function. The token is refreshed after every HTTP/HTTPS request that uses it.
- isAthlete will return a boolean value. Either True or False depends on the character information.

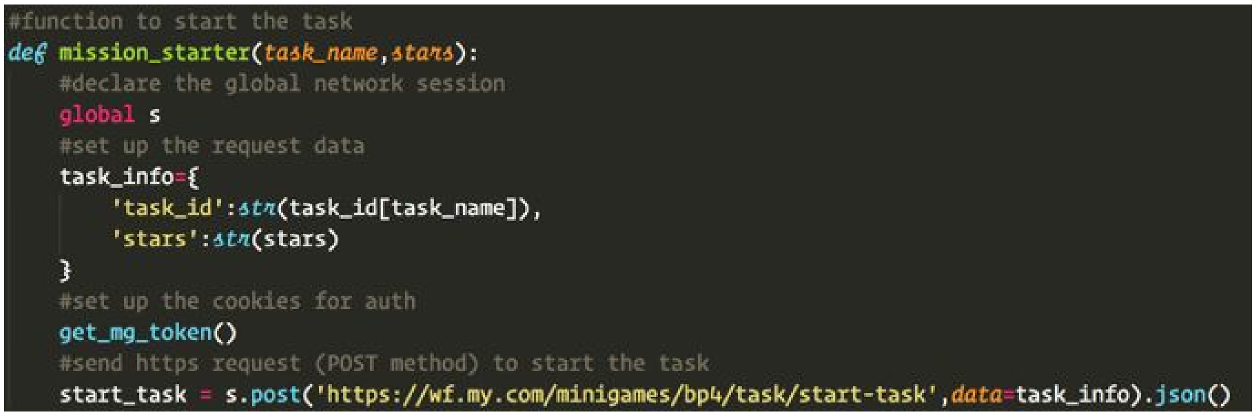
11. energy_refill function



```
#function to refill energy
def energy_refill():
    #declare the global network session
    global s
    #set up the cookies for auth
    get_mg_token()
    #send https request (POST method) to refill energy
    refill=s.post('https://wf.my.com/minigames/bp4/user/buy-energy')
```

- This function is called when character needs to refill the energy.
- Refilling energy requires the mg_token to authorize.
- The action will be triggered by using HTTP POST request to URL “https://wf.my.com/minigames/bp4/user/buy-energy”, no additional parameters needed.

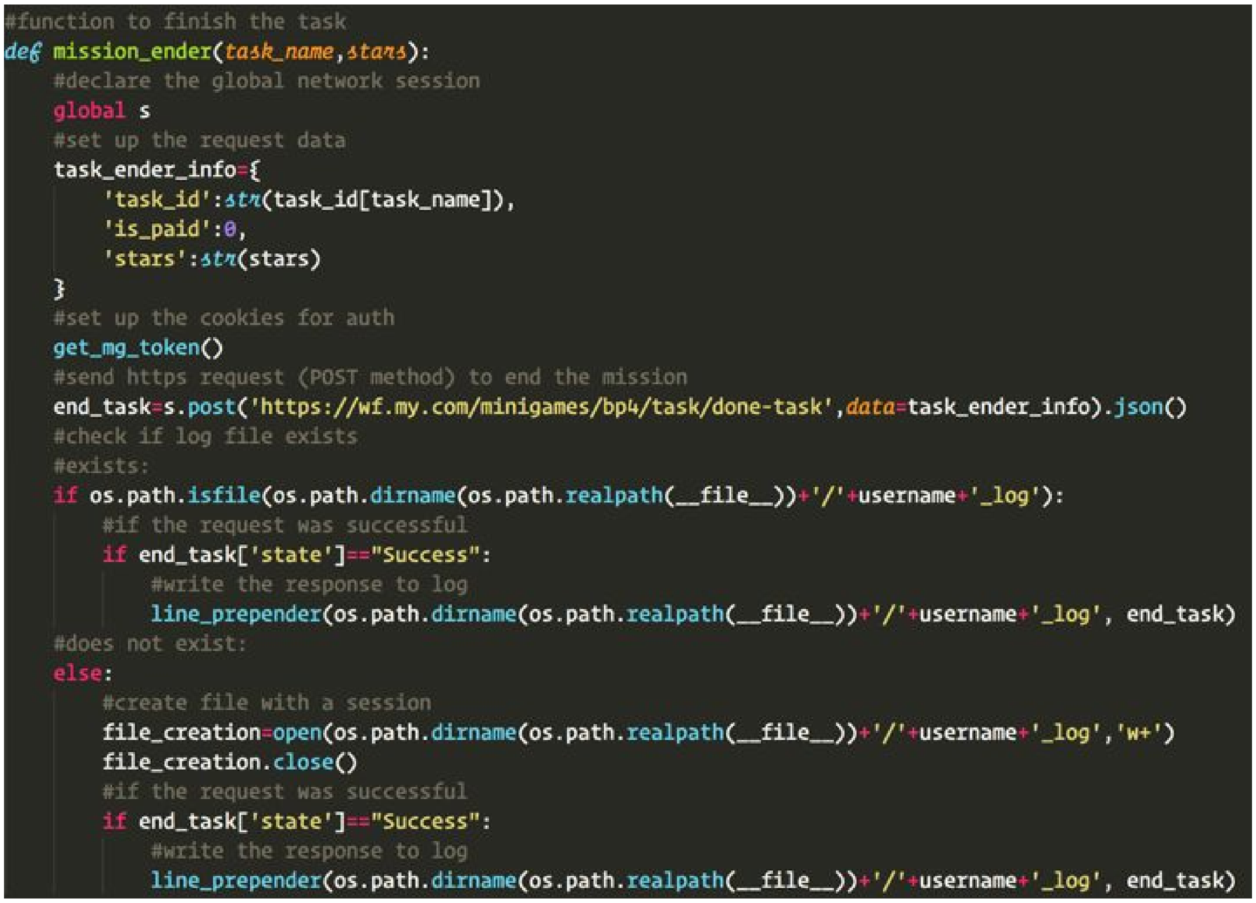
12. mission_starter function



```
#function to start the task
def mission_starter(task_name,stars):
    #declare the global network session
    global s
    #set up the request data
    task_info={
        'task_id':str(task_id[task_name]),
        'stars':str(stars)
    }
    #set up the cookies for auth
    get_mg_token()
    #send https request (POST method) to start the task
    start_task = s.post('https://wf.my.com/minigames/bp4/task/start-task',data=task_info).json()
```

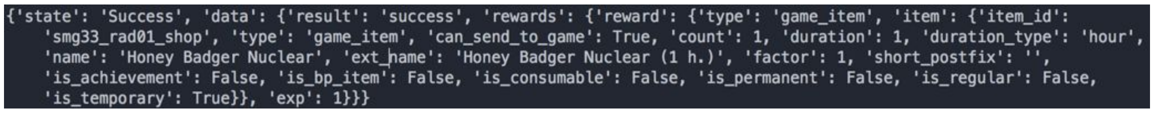
- This function is called to activate a mission.
- Activating a mission requires the mg_token to authorize.
- The action will be triggered by using HTTP POST request to URL “https://wf.my.com/minigames/bp4/task/start-task”.
- The task_id and stars will be sent as request data.

13. mission_ender function



```python
#function to finish the task
def mission_ender(task_name,stars):
    #declare the global network session
    global s
    #set up the request data
    task_ender_info={
        'task_id':str(task_id[task_name]),
        'is_paid':0,
        'stars':str(stars)
    }
    #set up the cookies for auth
    get_mg_token()
    #send https request (POST method) to end the mission
    end_task=s.post('https://wf.my.com/minigames/bp4/task/done-task',data=task_ender_info).json()
    #check if log file exists
    #exists:
    if os.path.isfile(os.path.dirname(os.path.realpath(__file__))+'/'+username+'_log'):
        #if the request was successful
        if end_task['state']=="Success":
            #write the response to log
            line_prepender(os.path.dirname(os.path.realpath(__file__))+'/'+username+'_log', end_task)
    #does not exist:
    else:
        #create file with a session
        file_creation=open(os.path.dirname(os.path.realpath(__file__))+'/'+username+'_log','w+')
        file_creation.close()
        #if the request was successful
        if end_task['state']=="Success":
            #write the response to log
            line_prepender(os.path.dirname(os.path.realpath(__file__))+'/'+username+'_log', end_task)
```

- This function is called to end a mission and write the reward into the unique log file for each user.
- Ending a mission requires the mg_token to authorize.
- The action will be triggered by using HTTP POST request to URL "https://wf.my.com/minigames/bp4/task/done-task".
- The task_id, is_paid and stars will be sent as request data.
- is_paid indicates if the user used Battle Points (a currency in the event) to speed up the mission progress. It is set to 0 as the script will not do so.
- Before writing the log file, os function is used to check if the log file exists already. When it does not exist, the program will create the file first.
- The log will only be written when the requests get a valid return.



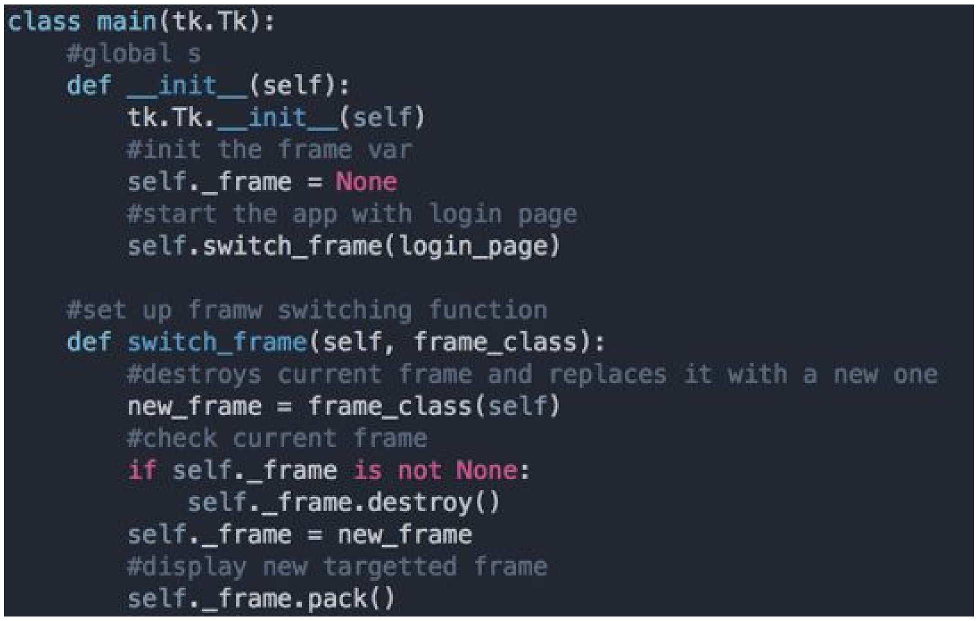
```
{'state': 'Success', 'data': {'result': 'success', 'rewards': {'reward': {'type': 'game_item', 'item': {'item_id':
    'smg33_rad01_shop', 'type': 'game_item', 'can_send_to_game': True, 'count': 1, 'duration': 1, 'duration_type': 'hour',
    'name': 'Honey Badger Nuclear', 'ext_name': 'Honey Badger Nuclear (1 h.)', 'factor': 1, 'short_postfix': '',
    'is_achievement': False, 'is_bp_item': False, 'is_consumable': False, 'is_permanent': False, 'is_regular': False,
    'is_temporary': True}}, 'exp': 1}}}
```

(Sample of Valid Return)

14. GUI

○ All the GUIs are based on python's TK framework interface, tkinter.

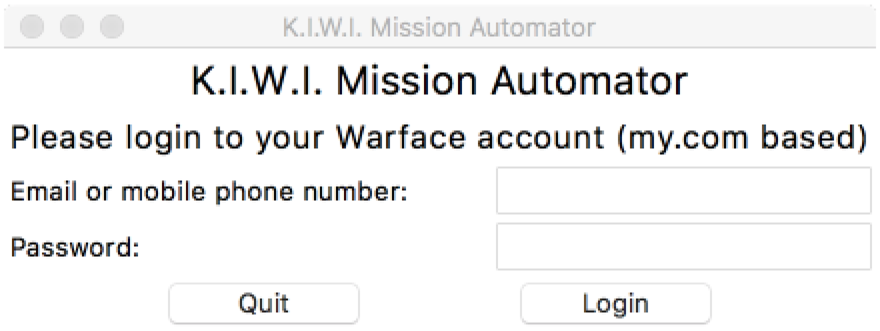
- main is inherited from tkinter's TK, to provide the base for the pages, and function, switch_frame.
- All the pages are frames inherited from tkinter's Frame.



```python
class main(tk.Tk):
    #global s
    def __init__(self):
        tk.Tk.__init__(self)
        #init the frame var
        self._frame = None
        #start the app with login page
        self.switch_frame(login_page)

    #set up framw switching function
    def switch_frame(self, frame_class):
        #destroys current frame and replaces it with a new one
        new_frame = frame_class(self)
        #check current frame
        if self._frame is not None:
            self._frame.destroy()
        self._frame = new_frame
        #display new targetted frame
        self._frame.pack()
```

15. Frame 1 - Login Page



K.I.W.I. Mission Automator

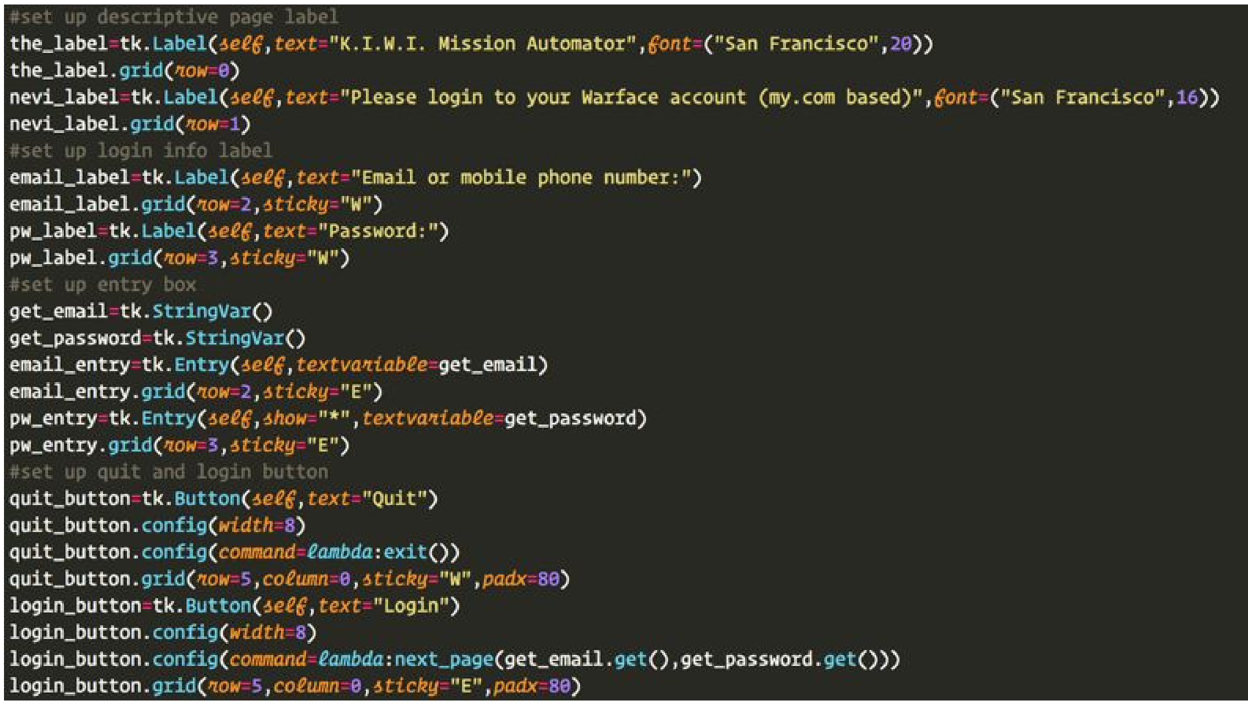
K.I.W.I. Mission Automator

Please login to your Warface account (my.com based)

Email or mobile phone number:

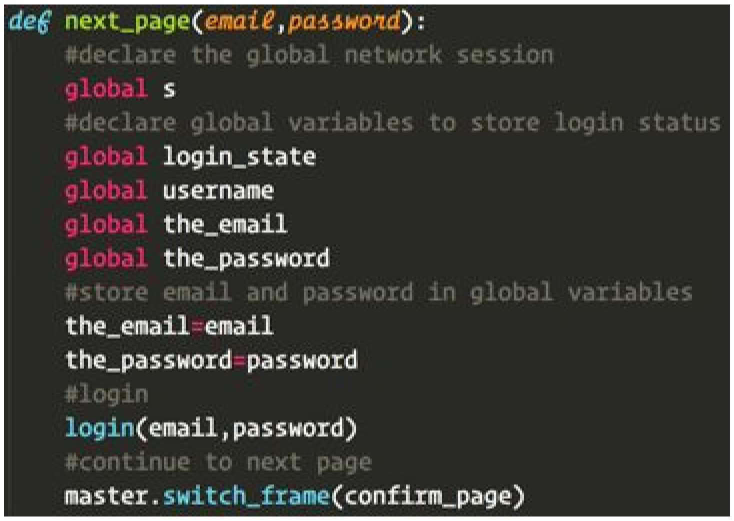
Password:

- The user will be asked to enter email or phone number that was used to register and the password to login to the game server.
- Instructive texts are Frame's label widgets
- Input fields are Frame's input widgets
- Quit and Login buttons are Frame's button widgets



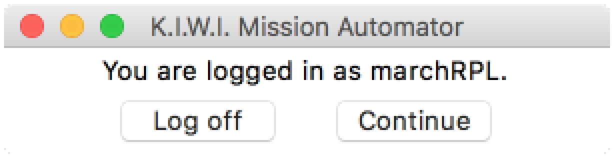
```
#set up descriptive page label
the_label=tk.Label(self,text="K.I.W.I. Mission Automator",font=("San Francisco",20))
the_label.grid(row=0)
nevi_label=tk.Label(self,text="Please login to your Warface account (my.com based)",font=("San Francisco",16))
nevi_label.grid(row=1)
#set up login info label
email_label=tk.Label(self,text="Email or mobile phone number:")
email_label.grid(row=2,sticky="W")
pw_label=tk.Label(self,text="Password:")
pw_label.grid(row=3,sticky="W")
#set up entry box
get_email=tk.StringVar()
get_password=tk.StringVar()
email_entry=tk.Entry(self,textvariable=get_email)
email_entry.grid(row=2,sticky="E")
pw_entry=tk.Entry(self,show="*",textvariable=get_password)
pw_entry.grid(row=3,sticky="E")
#set up quit and login button
quit_button=tk.Button(self,text="Quit")
quit_button.config(width=8)
quit_button.config(command=lambda:exit())
quit_button.grid(row=5,column=0,sticky="W",padx=80)
login_button=tk.Button(self,text="Login")
login_button.config(width=8)
login_button.config(command=lambda:next_page(get_email.get(),get_password.get()))
login_button.grid(row=5,column=0,sticky="E",padx=80)
```

- After clicking login, the program will send login information to the server and then continue to next page (Frame 2 - Confirmation Page)
- next_page function:

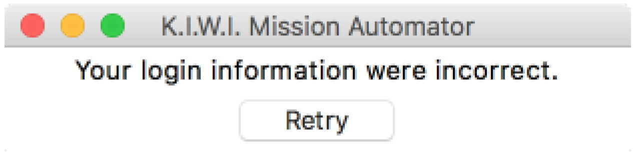


```
def next_page(email,password):
    #declare the global network session
    global s
    #declare global variables to store login status
    global login_state
    global username
    global the_email
    global the_password
    #store email and password in global variables
    the_email=email
    the_password=password
    #login
    login(email,password)
    #continue to next page
    master.switch_frame(confirm_page)
```

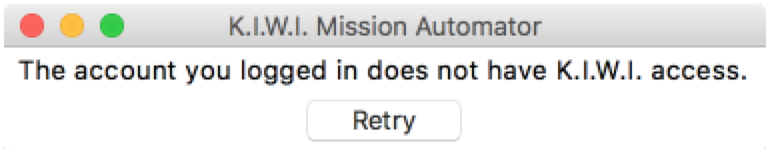
16. Frame 2 - Confirmation Page



You are logged in as marchRPL.

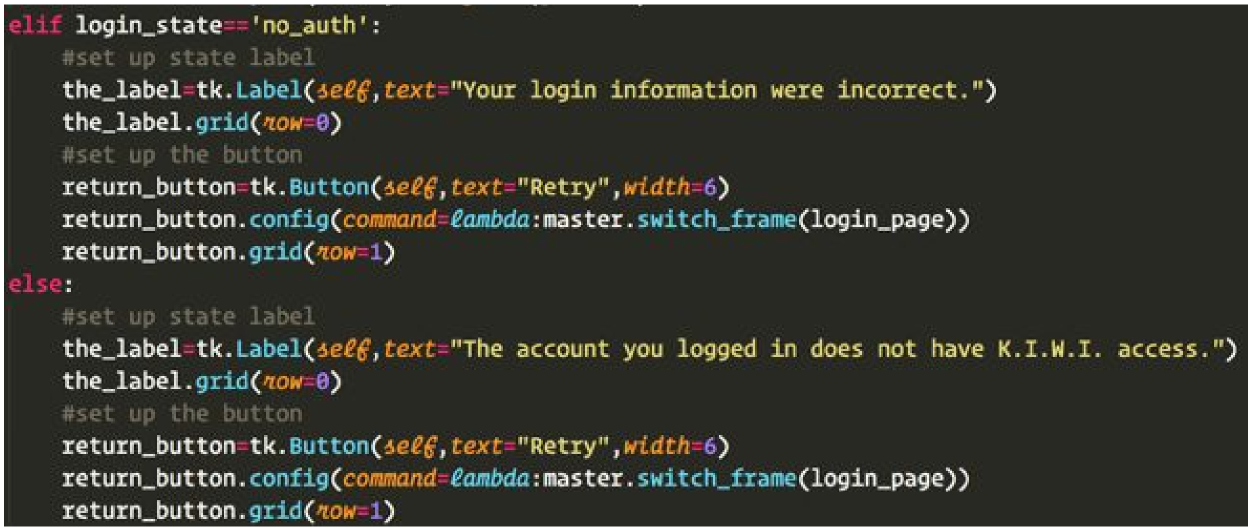

K.I.W.I. Mission Automator

Your login information were incorrect.


K.I.W.I. Mission Automator

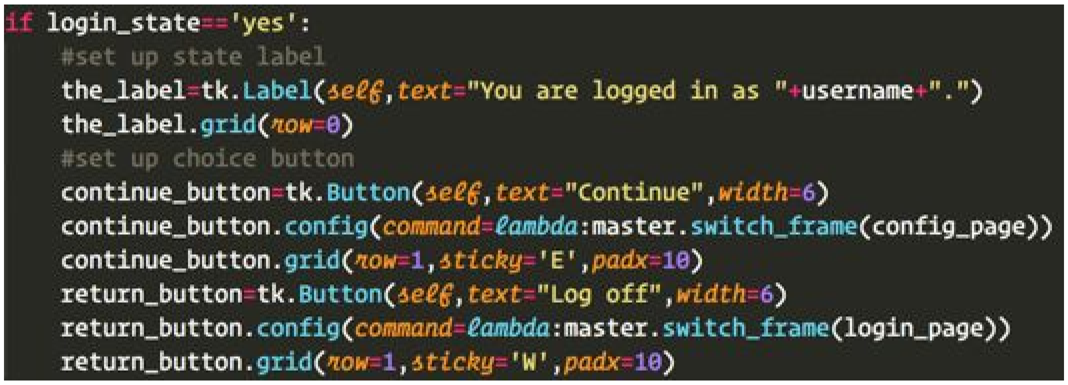
The account you logged in does not have K.I.W.I. access.

- When the login information is incorrect or the account does not have event access, the user will be instructed to enter another account. By tapping retry, the program will return to Frame 1 - Login Page.



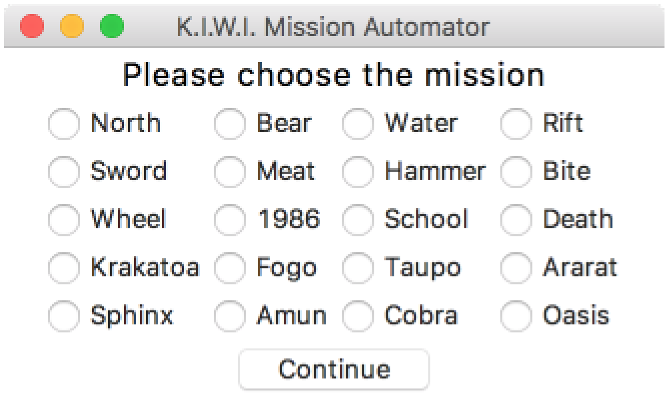
```
elif login_state=='no_auth':
    #set up state label
    the_label=tk.Label(self,text="Your login information were incorrect.")
    the_label.grid(row=0)
    #set up the button
    return_button=tk.Button(self,text="Retry",width=6)
    return_button.config(command=lambda:master.switch_frame(login_page))
    return_button.grid(row=1)
else:
    #set up state label
    the_label=tk.Label(self,text="The account you logged in does not have K.I.W.I. access.")
    the_label.grid(row=0)
    #set up the button
    return_button=tk.Button(self,text="Retry",width=6)
    return_button.config(command=lambda:master.switch_frame(login_page))
    return_button.grid(row=1)
```

- When the user logged into a valid account, by tapping continue, the program will switch to Frame 3 - Config Page.

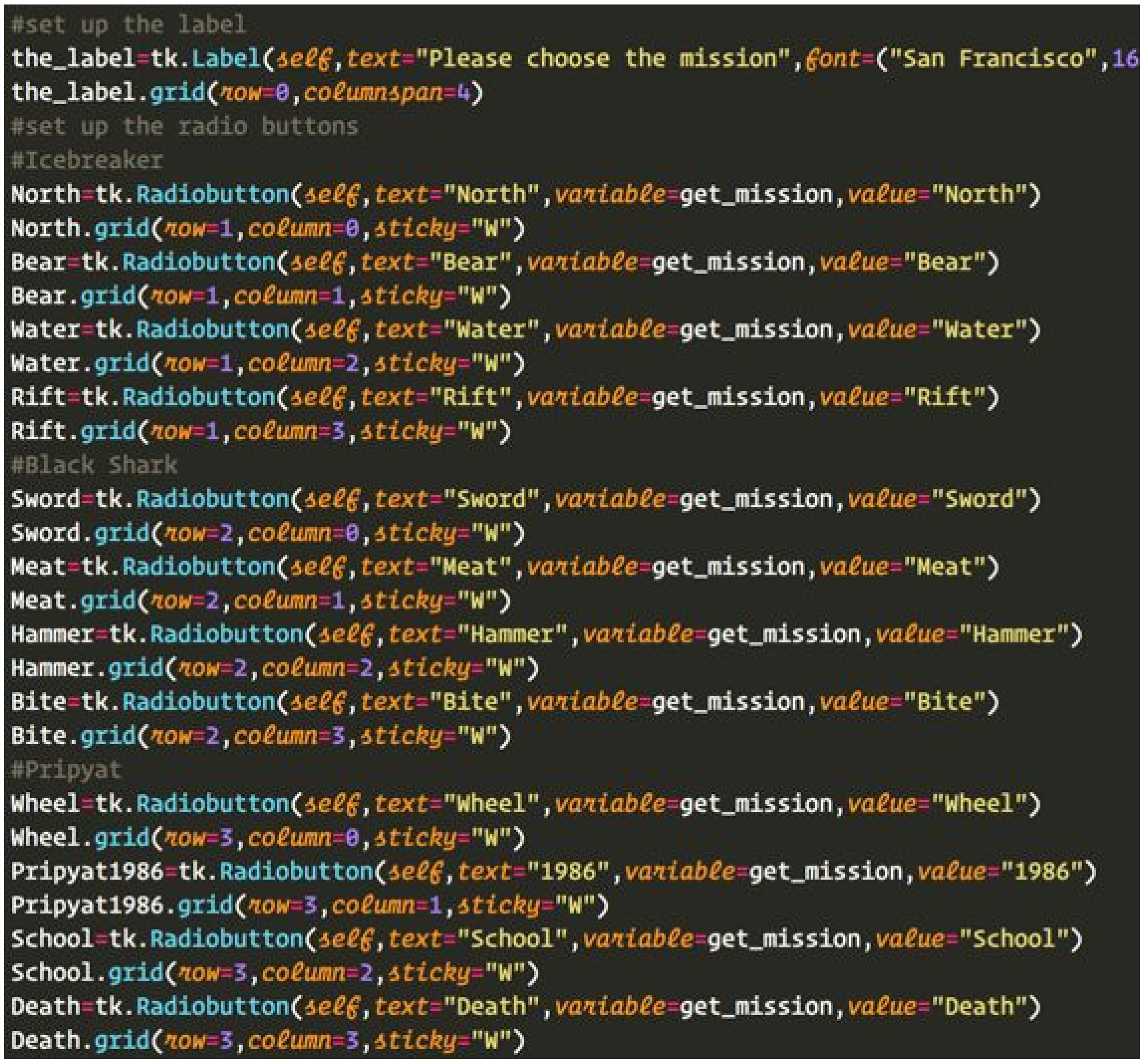


```
if login_state=='yes':
    #set up state label
    the_label=tk.Label(self,text="You are logged in as "+username+".")
    the_label.grid(row=0)
    #set up choice button
    continue_button=tk.Button(self,text="Continue",width=6)
    continue_button.config(command=lambda:master.switch_frame(config_page))
    continue_button.grid(row=1,sticky='E',padx=10)
    return_button=tk.Button(self,text="Log off",width=6)
    return_button.config(command=lambda:master.switch_frame(login_page))
    return_button.grid(row=1,sticky='W',padx=10)
```

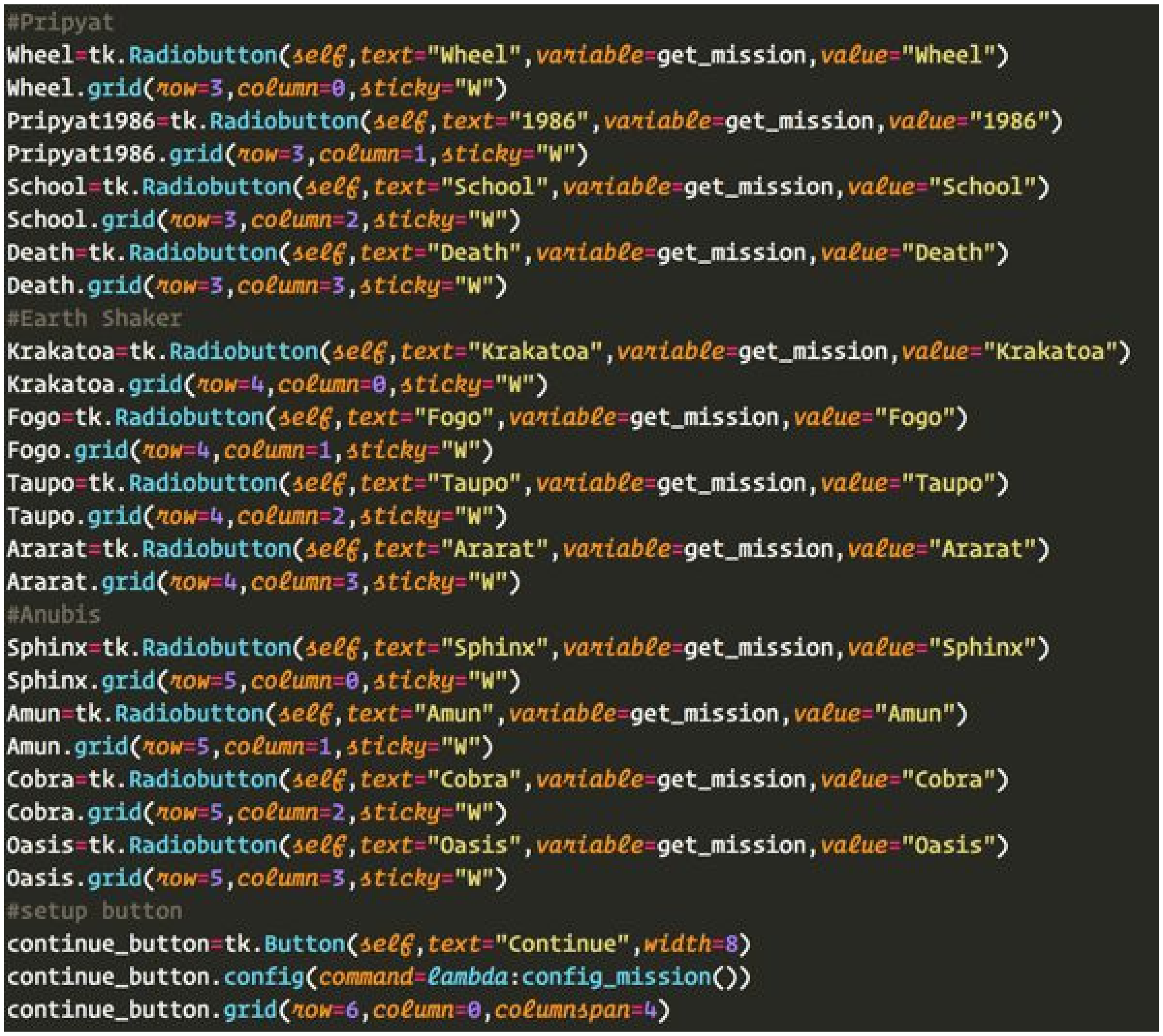
17. Frame 3 - Config Page



- This page uses Frame's radiobutton widget. The program will get the result of selection with StringVar.

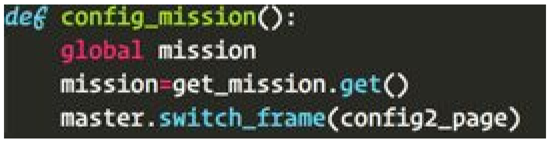


```
#set up the label
the_label=tk.Label(self,text="Please choose the mission",font=("San Francisco",16
the_label.grid(row=0,columnspan=4)
#set up the radio buttons
#Icebreaker
North=tk.Radiobutton(self,text="North",variable=get_mission,value="North")
North.grid(row=1,column=0,sticky="W")
Bear=tk.Radiobutton(self,text="Bear",variable=get_mission,value="Bear")
Bear.grid(row=1,column=1,sticky="W")
Water=tk.Radiobutton(self,text="Water",variable=get_mission,value="Water")
Water.grid(row=1,column=2,sticky="W")
Rift=tk.Radiobutton(self,text="Rift",variable=get_mission,value="Rift")
Rift.grid(row=1,column=3,sticky="W")
#Black Shark
Sword=tk.Radiobutton(self,text="Sword",variable=get_mission,value="Sword")
Sword.grid(row=2,column=0,sticky="W")
Meat=tk.Radiobutton(self,text="Meat",variable=get_mission,value="Meat")
Meat.grid(row=2,column=1,sticky="W")
Hammer=tk.Radiobutton(self,text="Hammer",variable=get_mission,value="Hammer")
Hammer.grid(row=2,column=2,sticky="W")
Bite=tk.Radiobutton(self,text="Bite",variable=get_mission,value="Bite")
Bite.grid(row=2,column=3,sticky="W")
#Pripyat
Wheel=tk.Radiobutton(self,text="Wheel",variable=get_mission,value="Wheel")
Wheel.grid(row=3,column=0,sticky="W")
Pripyat1986=tk.Radiobutton(self,text="1986",variable=get_mission,value="1986")
Pripyat1986.grid(row=3,column=1,sticky="W")
School=tk.Radiobutton(self,text="School",variable=get_mission,value="School")
School.grid(row=3,column=2,sticky="W")
Death=tk.Radiobutton(self,text="Death",variable=get_mission,value="Death")
Death.grid(row=3,column=3,sticky="W")
```



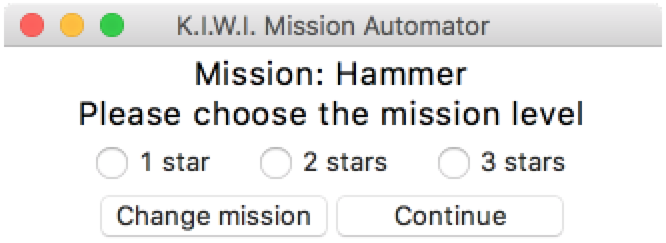
```
#Pripyat
Wheel=tk.Radiobutton(self,text="Wheel",variable=get_mission,value="Wheel")
Wheel.grid(row=3,column=0,sticky="W")
Pripyat1986=tk.Radiobutton(self,text="1986",variable=get_mission,value="1986")
Pripyat1986.grid(row=3,column=1,sticky="W")
School=tk.Radiobutton(self,text="School",variable=get_mission,value="School")
School.grid(row=3,column=2,sticky="W")
Death=tk.Radiobutton(self,text="Death",variable=get_mission,value="Death")
Death.grid(row=3,column=3,sticky="W")
#Earth Shaker
Krakatoa=tk.Radiobutton(self,text="Krakatoa",variable=get_mission,value="Krakatoa")
Krakatoa.grid(row=4,column=0,sticky="W")
Fogo=tk.Radiobutton(self,text="Fogo",variable=get_mission,value="Fogo")
Fogo.grid(row=4,column=1,sticky="W")
Taupo=tk.Radiobutton(self,text="Taupo",variable=get_mission,value="Taupo")
Taupo.grid(row=4,column=2,sticky="W")
Ararat=tk.Radiobutton(self,text="Ararat",variable=get_mission,value="Ararat")
Ararat.grid(row=4,column=3,sticky="W")
#Anubis
Sphinx=tk.Radiobutton(self,text="Sphinx",variable=get_mission,value="Sphinx")
Sphinx.grid(row=5,column=0,sticky="W")
Amun=tk.Radiobutton(self,text="Amun",variable=get_mission,value="Amun")
Amun.grid(row=5,column=1,sticky="W")
Cobra=tk.Radiobutton(self,text="Cobra",variable=get_mission,value="Cobra")
Cobra.grid(row=5,column=2,sticky="W")
Oasis=tk.Radiobutton(self,text="Oasis",variable=get_mission,value="Oasis")
Oasis.grid(row=5,column=3,sticky="W")
#setup button
continue_button=tk.Button(self,text="Continue",width=8)
continue_button.config(command=lambda:config_mission())
continue_button.grid(row=6,column=0,columnspan=4)
```

- config_mission function is defined in-class. It is used to store the result of user selection and switch to Frame 4 - Config Page 2



```
def config_mission():
    global mission
    mission=get_mission.get()
    master.switch_frame(config2_page)
```

18. Frame 4 - Config Page 2

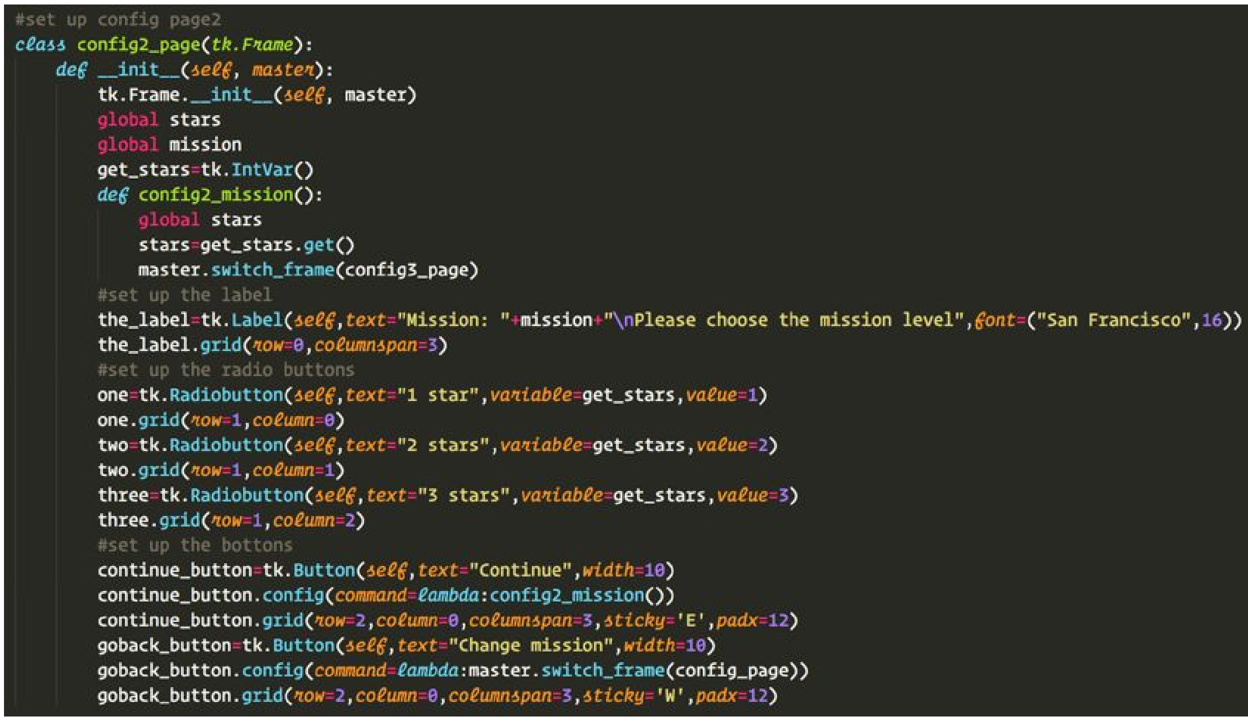


K.I.W.I. Mission Automator

Mission: Hammer

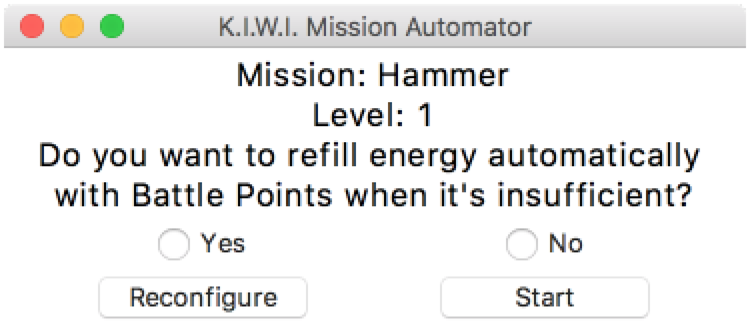
Please choose the mission level

- This frame allows the user to choose mission's difficulty or change the mission chosen. The techniques used is similar to the last frame.



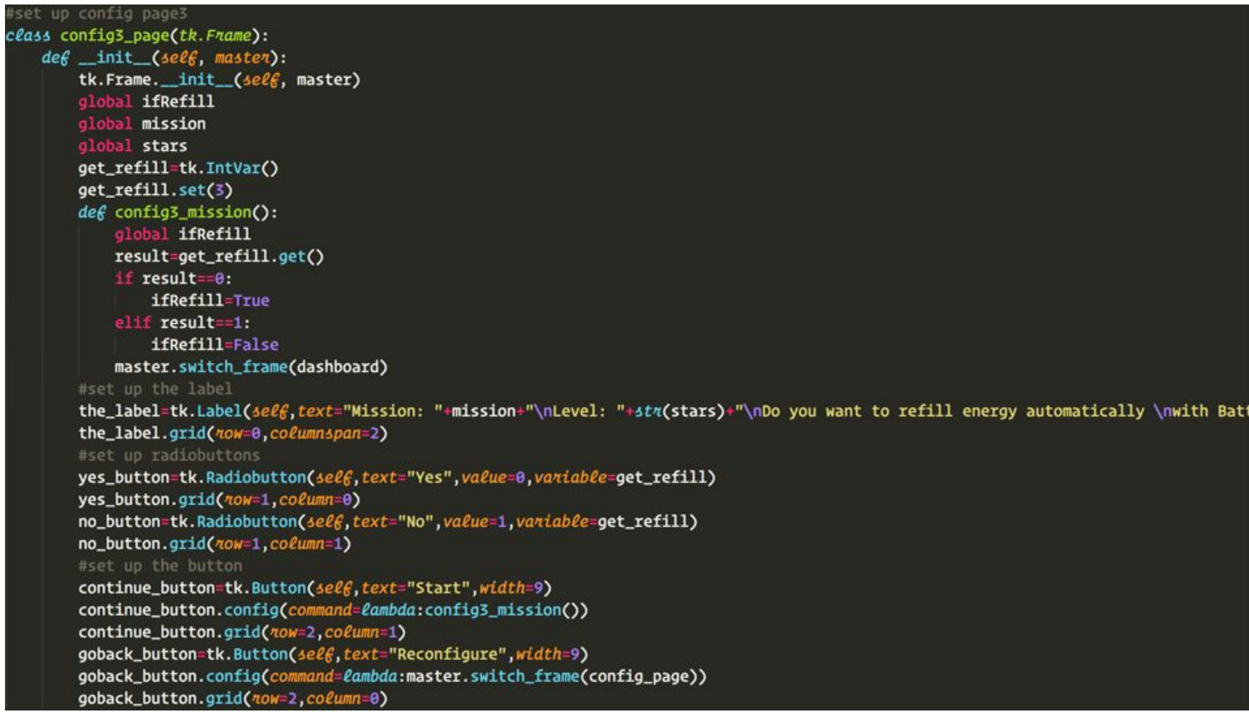
```
#set up config page2
class config2_page(tk.Frame):
    def __init__(self, master):
        tk.Frame.__init__(self, master)
        global stars
        global mission
        get_stars=tk.IntVar()
        def config2_mission():
            global stars
            stars=get_stars.get()
            master.switch_frame(config3_page)
        #set up the label
        the_label=tk.Label(self,text="Mission: "+mission+"\nPlease choose the mission level",font=("San Francisco",16))
        the_label.grid(row=0,columnspan=3)
        #set up the radio buttons
        one=tk.Radiobutton(self,text="1 star",variable=get_stars,value=1)
        one.grid(row=1,column=0)
        two=tk.Radiobutton(self,text="2 stars",variable=get_stars,value=2)
        two.grid(row=1,column=1)
        three=tk.Radiobutton(self,text="3 stars",variable=get_stars,value=3)
        three.grid(row=1,column=2)
        #set up the bottons
        continue_button=tk.Button(self,text="Continue",width=10)
        continue_button.config(command=lambda:config2_mission())
        continue_button.grid(row=2,column=0,columnspan=3,sticky='E',padx=12)
        goback_button=tk.Button(self,text="Change mission",width=10)
        goback_button.config(command=lambda:master.switch_frame(config_page))
        goback_button.grid(row=2,column=0,columnspan=3,sticky='W',padx=12)
```

19. Frame 5 - Config Page 3



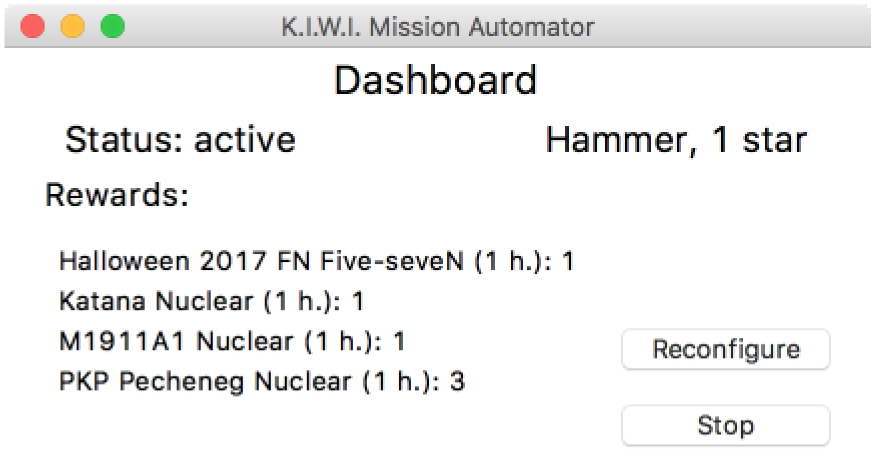
Mission: Hammer
Level: 1
Do you want to refill energy automatically
with Battle Points when it's insufficient?

- This frame allows user to choose if they want to refill the energy of event character automatically. They can either choose to reconfigure the mission or start the automator. Techniques used are similar to last two frames.



```
#set up config page3
class config3_page(tk.Frame):
    def __init__(self, master):
        tk.Frame.__init__(self, master)
        global ifRefill
        global mission
        global stars
        get_refill=tk.IntVar()
        get_refill.set(3)
        def config3_mission():
            global ifRefill
            result=get_refill.get()
            if result==0:
                ifRefill=True
            elif result==1:
                ifRefill=False
            master.switch_frame(dashboard)
        #set up the label
        the_label=tk.Label(self,text="Mission: "+mission+"\nLevel: "+str(stars)+"\nDo you want to refill energy automatically \nwith Batt
        the_label.grid(row=0,columnspan=2)
        #set up radiobuttons
        yes_button=tk.Radiobutton(self,text="Yes",value=0,variable=get_refill)
        yes_button.grid(row=1,column=0)
        no_button=tk.Radiobutton(self,text="No",value=1,variable=get_refill)
        no_button.grid(row=1,column=1)
        #set up the button
        continue_button=tk.Button(self,text="Start",width=9)
        continue_button.config(command=lambda:config3_mission())
        continue_button.grid(row=2,column=1)
        goback_button=tk.Button(self,text="Reconfigure",width=9)
        goback_button.config(command=lambda:master.switch_frame(config_page))
        goback_button.grid(row=2,column=0)
```

20. Frame 6 - Dashboard



Dashboard

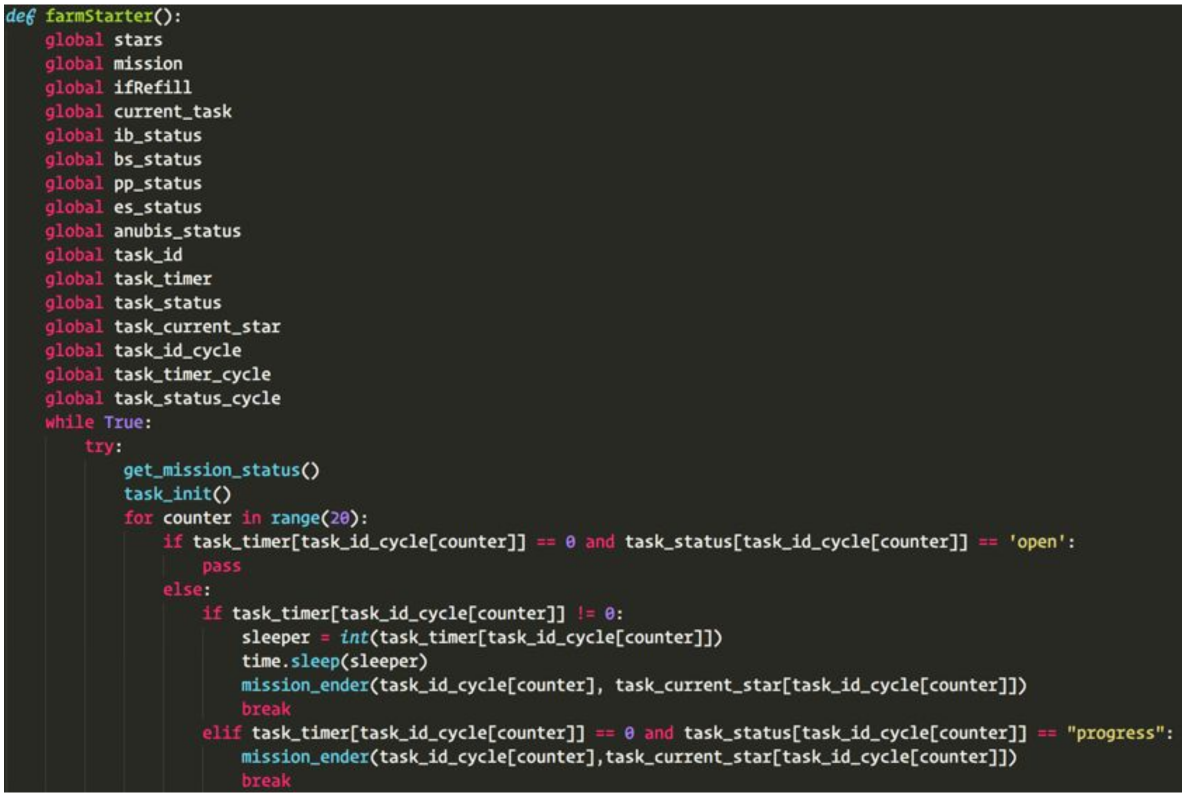
Status: active　　Hammer, 1 star

Rewards:

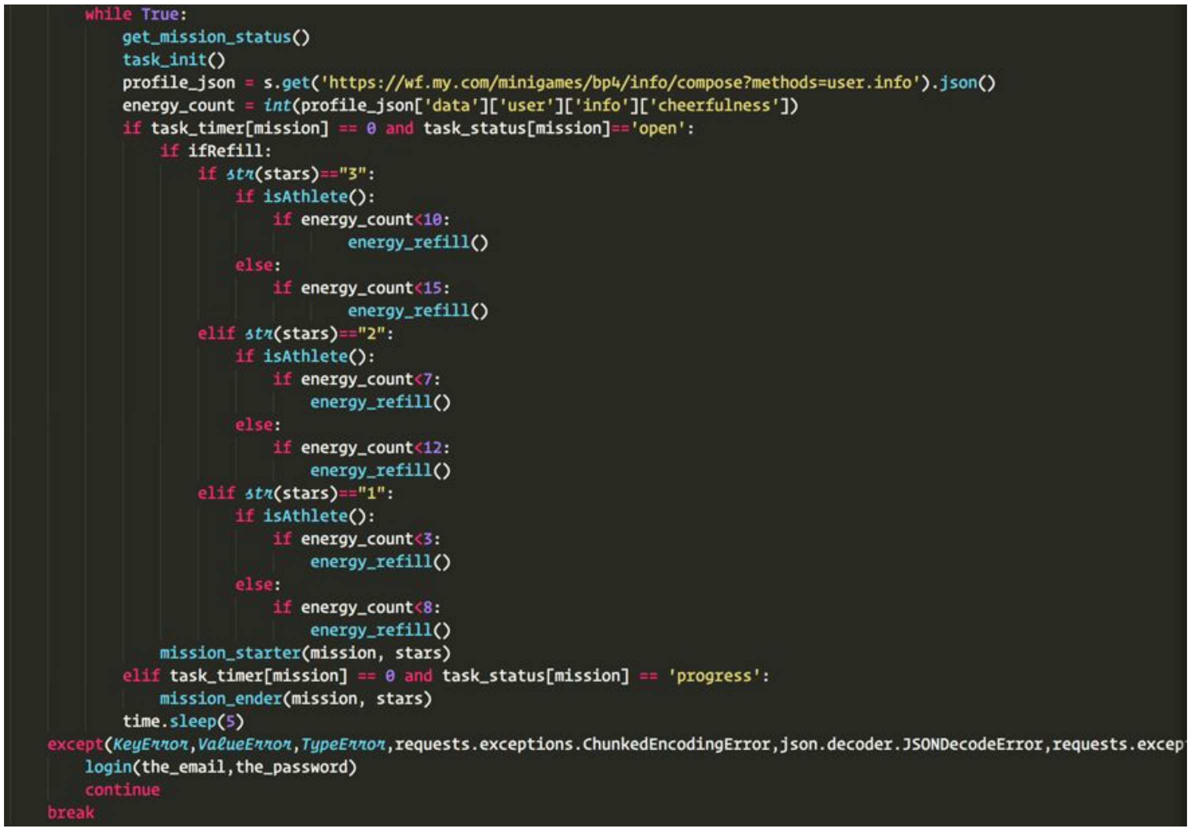
Halloween 2017 FN Five-seveN (1 h.): 1
Katana Nuclear (1 h.): 1
M1911A1 Nuclear (1 h.): 1
PKP Pecheneg Nuclear (1 h.): 3

- The dashboard will allow the user to see the rewards they get from the event, however, only the rewards received via the automator will shows.
- The dashboard also allows user to stop the automator or reconfigure. By tapping reconfigure, the program will return to Frame 3 - Config Page.
- The reward list is written on a canvas.
- To avoid the freeze of GUI, the farming function is run in another process with multiprocessing.

- Farming function is named farmStarter. The function will first check if there’s currently an active mission. If so, the farm will start after it’s finished.

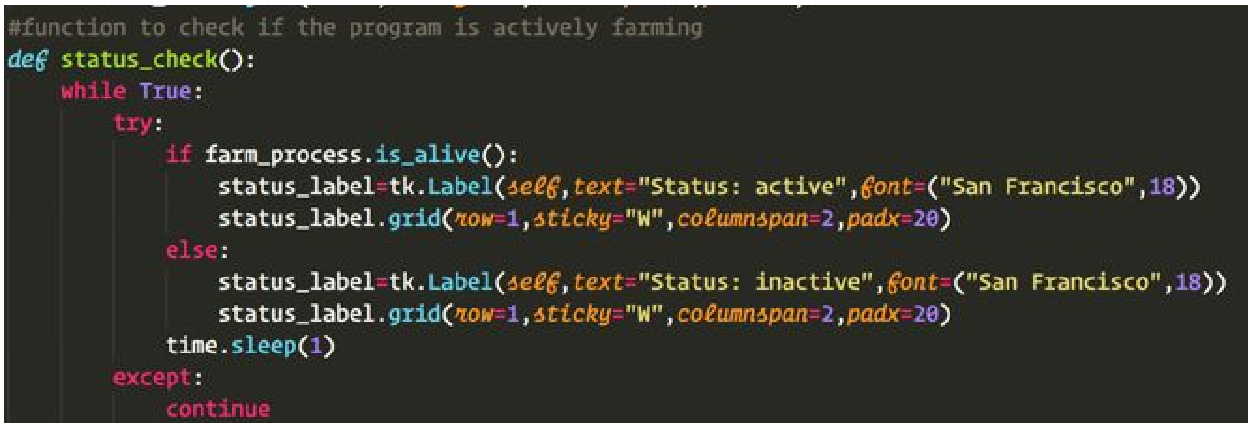


```
def farmStarter():
    global stars
    global mission
    global ifRefill
    global current_task
    global ib_status
    global bs_status
    global pp_status
    global es_status
    global anubis_status
    global task_id
    global task_timer
    global task_status
    global task_current_star
    global task_id_cycle
    global task_timer_cycle
    global task_status_cycle
    while True:
        try:
            get_mission_status()
            task_init()
            for counter in range(20):
                if task_timer[task_id_cycle[counter]] == 0 and task_status[task_id_cycle[counter]] == 'open':
                    pass
                else:
                    if task_timer[task_id_cycle[counter]] != 0:
                        sleeper = int(task_timer[task_id_cycle[counter]])
                        time.sleep(sleeper)
                        mission_ender(task_id_cycle[counter], task_current_star[task_id_cycle[counter]])
                        break
                    elif task_timer[task_id_cycle[counter]] == 0 and task_status[task_id_cycle[counter]] == "progress":
                        mission_ender(task_id_cycle[counter],task_current_star[task_id_cycle[counter]])
                        break
```

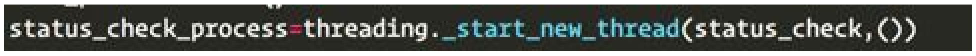


```
        while True:
            get_mission_status()
            task_init()
            profile_json = s.get('https://wf.my.com/minigames/bp4/info/compose?methods=user.info').json()
            energy_count = int(profile_json['data']['user']['info']['cheerfulness'])
            if task_timer[mission] == 0 and task_status[mission]=='open':
                if ifRefill:
                    if str(stars)=="3":
                        if isAthlete():
                            if energy_count<10:
                                    energy_refill()
                        else:
                            if energy_count<15:
                                    energy_refill()
                    elif str(stars)=="2":
                        if isAthlete():
                            if energy_count<7:
                                energy_refill()
                        else:
                            if energy_count<12:
                                energy_refill()
                    elif str(stars)=="1":
                        if isAthlete():
                            if energy_count<3:
                                energy_refill()
                        else:
                            if energy_count<8:
                                energy_refill()
                mission_starter(mission, stars)
            elif task_timer[mission] == 0 and task_status[mission] == 'progress':
                mission_ender(mission, stars)
            time.sleep(5)
    except(KeyError,ValueError,TypeError,requests.exceptions.ChunkedEncodingError,json.decoder.JSONDecodeError,requests.excep
        login(the_email,the_password)
        continue
    break
```

- Similar to the login function explained in before, farmStarter utilizes a while Loop with except to avoid login session expires.
- There’s also a thread to check whether the farmStarter’s process is still active.



```
#function to check if the program is actively farming
def status_check():
    while True:
        try:
            if farm_process.is_alive():
                status_label=tk.Label(self,text="Status: active",font=("San Francisco",18))
                status_label.grid(row=1,sticky="W",columnspan=2,padx=20)
            else:
                status_label=tk.Label(self,text="Status: inactive",font=("San Francisco",18))
                status_label.grid(row=1,sticky="W",columnspan=2,padx=20)
            time.sleep(1)
        except:
            continue
```



```
status_check_process=threading._start_new_thread(status_check,())
```

Word Counts: 1267