Criterion C: Development

Throughout this document, screenshots of code will be heavily commented to assist in explaining the function of all methods and classes.

Libraries Imported

```
import calendar
import datetime
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk

import gspread
from oauth2client.service_account import ServiceAccountCredentials
```

Fig. 1 - Imports

Calendar and Datetime are used for generating and reading timestamps.

TkInter is a package used to create GUIs. It was chosen for its extensive documentations, and for my previous experience using the module in class

GSpread provides Python bindings for Google Sheets API v4. Allows interacting with and creating spreadsheets. I used Google Sheets because it was easily structured and allowed for remote access from any internet connected computer.

OAuth2Client allows for generating Google API credentials from JSON files, authorizes GSpread requests.

NOTE : These libraries are not all used in every file, further in this document, imports will be specified for each file

All Classes

Name	Parent	Purpose
KTrackerApp	None	Manages menubar, contains all pages, manages switching pages.
HomeScreen	KTrackerApp	Displays all activities, manages what week of activities is displayed.
Day	HomeScreen	Organizes ActivityBoxes on the HomeScreen.
ActivityBox	Day	Presents data for an activity.
ActivityData	None	Stores data for an activity.
AddActivityScreen	KTrackerApp	Input for new activities.
ProfileScreen	KTrackerApp	Displays statistics, initializes ChangePasswordWindow.
Authorization	None	Manages LoginScreen and SignUpScreen, destroyed and initializes KTrackerApp when completed.
LoginScreen	Authorization	Allows user to login, verifies credentials.
SignUpScreen	Authorization	Allows user to sign up, adds new users to database.
ChangePasswordWindow	None	Allows a user to change their password.
EditActivityWindow	None	Opens window allowing the user to

		edit an activity. Initialized by a ActivityBox.

List of Techniques

User Interface

TkInter

The approach I took with TkInter relied heavily on classes. The main application is it's own class, containing all other pages, which are also classes. Many elements of the GUI are also classes, such as ActivityBoxes. One benefit of this approach is that it allows classes to have overlapping variable names without causing any difficulties.

How Data is Stored

I chose to use Google Sheets to create a quick and easy to use database. Through Google's Google Sheets API and the GSpread module, I can easily access and edit the database. Individual users have separate worksheets named matching their username. Another advantage of Google Sheets is that the data is easily viewable and understood without the use of the program, this is ideal for bug testing.

	A	B	C	D	E	F	G
1	Password						
2							
3	STATISTICS				Averages		
4		Distance	Time	Count	Distance	Time	
5	TOTAL	0	0	0	0	0	
6	SKI	0	0	0	0	0	
7	BIKE	0	0	0	0	0	
8	RUN	0	0	0	0	0	
9							
10	ACTIVITIES						
11	Sport	Title	Time	Distance	Description	ID	
12							

Fig. 2 - Blank template for user worksheet

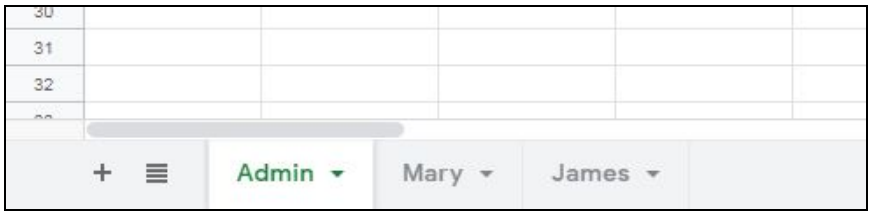

Fig. 3 - View of worksheets

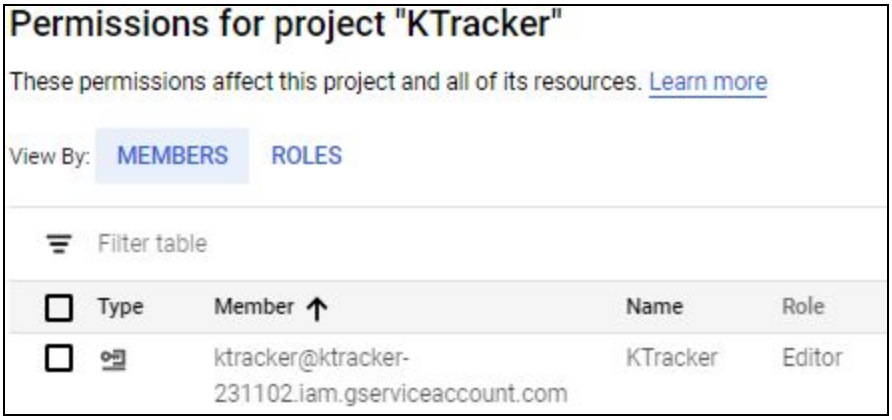

Fig. 4 - View of Google Sheets API manager showing permissions granted to KTracker project

Primary Methods and Classes

Main

Contains KTrackerApp, as well as the code for starting the program. The main KTrackerApp is not initialized until the user has been authenticated.

```
import tkinter as tk

import AddActivityScreen
import HomeScreen
import ProfileScreen
import auth
```

Fig. 5 - main.py imports

```
def boot_app():
    # Called by the Login or SignUp methods once the user is authorized
    # Creates an instance of KTrackerApp
    app = KTrackerApp()
    app.mainloop()
```

Fig. 6 - boot_app method

```
if __name__ == "__main__":
    # Creates an instance of Authorization
    auth = auth.Authorization('')
    auth.geometry("360x360")
    auth.mainloop()
```

Fig. 7 - First lines ran, starts authorization process

Authorization

Used only for login and signup, destroyed when user is authorized. One difficulty that arose when writing this code involved calling functions with TkInter buttons. When I attempted to pass arguments into the functions, they would run as soon as the program was started, then do nothing when the button was pressed. I solved this by calling them as lambda functions, which prevented them from running instantaneously.

```
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk

import KTrackerApp
import SheetTools
```

Fig. 8 - auth.py imports

```python
class Authorization(tk.Tk):

    def __init__(self, admin_key):

        # Initializes and names the window
        tk.Tk.__init__(self)
        tk.Tk.wm_title(self, string="Authorizer")

        # Used for debugging, skips login process and enters as user 'Admin'
        if admin_key == '10267':
            main.currentUser = 'Admin'
            self.destroy()
            main.boot_app()
            quit()

        # Creates container for pages
        self.container = tk.Frame(self)
        self.container.pack(side="top", fill="both", expand=True)
        self.container.grid_rowconfigure(0, weight=1)
        self.container.grid_columnconfigure(0, weight=1)

        self.frames = {}

        # Creates instances of LoginScreen and SignUpScreen
        for F in (LoginScreen, SignUpScreen):
            frame = F(self.container, self)

            self.frames[F] = frame

            # Places them in the window
            frame.grid(column=0, row=0, sticky="nsew")

        # Opens the Login screen
        self.switch_frame(1)
```

Fig. 9 - Master of Login and Signup screens

```python
def switch_frame(self, page):  # Manages switching pages
    # If the page is 1, open login, else open signup
    if page == 1:
        frame = LoginScreen(self.container, self)
    else:
        frame = SignUpScreen(self.container, self)
    # Selects the username entry box and sets up the page
    frame.username_input.focus()
    frame.grid(column=0, row=0, sticky="nsew")
    frame.tkraise()
    tk.Tk.wm_title(
        self,
        string=("KTracker | " + frame.name)
    )
```

Fig. 10 - Switch_frame method of Authorization Class

```
class LoginScreen(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        # Fonts used in this page
        self.fonts = {
            'TitleText': ('Roboto', '25', 'bold')
        }

        self.configure(bg='#f5f5fa')
        self.controller = controller
        self.name = 'Login'

        # Initialize Widgets
        self.login_label = tk.Label(self, text='Login', font=self.fonts['TitleText'], bg='#f5f5fa')
        self.username_label = tk.Label(self, text='Username : ', bg='#f5f5fa')
        self.password_label = tk.Label(self, text='Password : ', bg='#f5f5fa')

        self.username_input = tk.Entry(self)
        self.password_input = tk.Entry(self, show="\u2022")

        self.submit = ttk.Button(self, text='Submit', command=lambda: self.login())
        self.signup = ttk.Button(self, text='Sign Up', command=lambda: self.switch_mode())

        # Place Widgets
        self.login_label.grid(column=0, row=0, padx=10, pady=10)
        self.username_label.grid(column=0, row=1)
        self.password_label.grid(column=0, row=2)

        self.username_input.grid(column=1, row=1)
        self.password_input.grid(column=1, row=2)

        self.submit.grid(column=0, row=3, padx=5, pady=5)
        self.signup.grid(column=0, row=4, padx=5, pady=5)
```

Fig. 11 - LoginScreen class

```python
def login(self):
    # Retrieves username and password from input fields
    username = self.username_input.get()
    password = self.password_input.get()

    # Ensures that there is input
    if not username or not password:
        return False

    try:  # Fails if user does not exist
        SheetTools.get_user_data(username)

        # If the user and password match, start the main app and close window
        if SheetTools.get_password(username) == password:
            main.currentUser = username
            self.controller.destroy()
            main.boot_app()
            return True
        else:  # Shows an error popup
            tk.messagebox.showerror(
                'Error',
                'Sorry, unrecognized username or password.'
            )
            return False
    except ValueError:  # Shows an error popup
        tk.messagebox.showerror(
            'Error',
            'Sorry, unrecognized username or password.'
        )
        return False
```

Fig. 12 - Login method of LoginScreen class

```python
class SignUpScreen(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        # Fonts used in this page
        self.fonts = {
            'TitleText': ('Roboto', '25', 'bold')
        }

        self.configure(bg='#f5f5fa')
        self.controller = controller
        self.name = 'Sign Up'

        # Declare Widgets
        self.sign_up_label = tk.Label(self, text='Sign Up', font=self.fonts['TitleText'], bg='#f5f5fa')
        self.username_label = tk.Label(self, text='Username : ', bg='#f5f5fa')
        self.password1_label = tk.Label(self, text='Password : ', bg='#f5f5fa')
        self.password2_label = tk.Label(self, text='Confirm : ', bg='#f5f5fa')

        self.username_input = tk.Entry(self)
        self.password1_input = tk.Entry(self, show="\u2022")
        self.password2_input = tk.Entry(self, show="\u2022")

        self.submit = ttk.Button(self, text='Submit', command=lambda: self.add_user())
        self.signup = ttk.Button(self, text='Login', command=lambda: self.switch_mode())

        # Arrange Widgets
        self.sign_up_label.grid(column=0, row=0, padx=10, pady=10)
        self.username_label.grid(column=0, row=1)
        self.password1_label.grid(column=0, row=2)
        self.password2_label.grid(column=0, row=3)

        self.username_input.grid(column=1, row=1)
        self.password1_input.grid(column=1, row=2)
        self.password2_input.grid(column=1, row=3)

        self.submit.grid(column=0, row=4, padx=5, pady=5)
        self.signup.grid(column=0, row=5, padx=5, pady=5)
```

Fig. 13 - SignUpScreen Class

```python
def add_user(self):
    # Retrieves username and password from inputs
    username = self.username_input.get()
    password1 = self.password1_input.get()
    password2 = self.password2_input.get()

    # Ensures there has been input
    if not username or not password1 or not password2:
        return False

    # Returns 1 if user exists, 2 if successful, 3 if passwords don't match
    result = SheetTools.add_user(username, password1, password2)

    if result == 1:
        # Notifies user that user already exists
        tk.messagebox.showerror(
            'Error',
            'Sorry, user already exists.'
        )
        return False

    elif result == 2:
        # Destroys the window and starts the main app
        self.controller.destroy()
        main.boot_app()
        return True

    elif result == 3:
        # Notifies the user that the passwords don't match
        tk.messagebox.showerror(
            'Error',
            'Sorry, passwords don\'t match'
        )
        return False
```

Fig. 14 - Add user method of SignUpScreen class

KTrackerApp

This is the main class which all contains all pages.

```
import tkinter as tk

import AddActivityScreen
import HomeScreen
import ProfileScreen
import auth
```

Fig. 15 - main.py imports

```python
class KTrackerApp(tk.Tk):

    def __init__(self, *args, **kwargs):

        # Initializes and sets up window
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, string="KTracker")
        self.geometry('1440x380')

        # Sets up container (all pages are placed in this frame)
        self.container = tk.Frame(self)
        self.container.pack(side="top", fill="both", expand=True)
        self.container.grid_rowconfigure(0, weight=1)
        self.container.grid_columnconfigure(0, weight=1)

        # Creates menubar, adds a button for each page
        self.menubar = tk.Menu(self.container)
        self.menubar.add_command(label="Exit", command=quit)
        self.menubar.add_command(
            label="Home",
            command=lambda: self.show_frame(HomeScreen.HomeScreen)
        )
        self.menubar.add_command(
            label="Profile",
            command=lambda: self.show_frame(ProfileScreen.ProfileScreen)
        )
        self.menubar.add_command(
            label="Add Activity",
            command=lambda: self.show_frame(AddActivityScreen.AddActivityScreen)
        )
        tk.Tk.config(self, menu=self.menubar)

        self.frames = {}

        self.pages = [  # List of all page objects
            HomeScreen.HomeScreen,
            ProfileScreen.ProfileScreen,
            AddActivityScreen.AddActivityScreen
        ]

        # Creates all pages, places them in container frame
        for F in self.pages:
            frame = F(self.container, self)
            frame.grid(column=0, row=0, sticky='nsew')

        # Opens the home page
        self.show_frame(HomeScreen.HomeScreen)
```

Fig. 16 - KTrackerApp initialization

```
def switch_frame(self, f):

    # If the frame doesn't exist, create it
    if f in self.frames:
        frame = self.frames[f]
    else:
        frame = f(self.container, self)
        self.frames[f] = frame
        frame.grid(column=0, row=0, sticky='nsew')

    # Bring the frame to the front and name window
    frame.tkraise()
    tk.Tk.wm_title(
        self,
        string=("KTracker | " + frame.name)
    )
```

Fig. 17 - Switch_frame method of KTrackerApp class

HomeScreen

Displays a user's activities by week. I initially believed it would be impractical to display more than one activity at a time, due to the amount of code required to display each one. This was solved by displaying the activities as the ActivityBox class, which allowed as many activities to be displayed as needed.

```
import calendar
import tkinter as tk
from tkinter import ttk

import DayObject
import ActivityBoxObject
import KTrackerApp
import StatisticsTools
```

Fig. 18 - HomeScreen.py imports

```
class HomeScreen(tk.Frame):

    def __init__(self, parent, controller):

        # Creates and sets up frame
        tk.Frame.__init__(self, parent)
        self.controller = controller
        self.name = 'Home'

        # Used for storing and naming Day frames
        self.days = []
        self.day_names = [
            'Monday', 'Tuesday', 'Wednesday', 'Thursday',
            'Friday', 'Saturday', 'Sunday']

        self.current_week = 0

        # Initialize Widgets
        refresh_button = ttk.Button(self, text='Refresh', command=lambda: self.refresh(0))
        last_week_button = ttk.Button(self, text='Last Week', command=lambda: self.refresh(-1))
        next_week_button = ttk.Button(self, text='Next Week', command=lambda: self.refresh(1))

        # Place Widgets
        refresh_button.grid(column=0, row=0, sticky='w')
        last_week_button.grid(column=12, row=0, sticky='w')
        next_week_button.grid(column=13, row=0, sticky='w')

        # Creates a Day frame for each day of the week
        for d in range(7):
            self.days.append(
                DayObject.Day(self, d, self.day_names[d])
            )

        # Loads activities into Day frames
        self.refresh(0)
```

Fig. 19 - HomeScreen class

```
def refresh(self, offset):

    # Adjusts the offset based on button pressed
    self.current_week += offset

    # Wipes previous activities
    for d in self.days:
        for widget in d.winfo_children():
            widget.destroy()

    # Sets up Day frames
    for d in self.days:
        d.grid(column=d.day*2, columnspan=2, row=1, sticky='n')
        d.grid_rowconfigure(0, weight=1)
        d.grid_columnconfigure(0, weight=1)
        d.title = tk.Label(d, text=d.day_name)
        d.title.grid(column=0, row=0)

    # Gets the user's activities from the current week
    activities = StatisticsTools.get_this_week(
        KTrackerApp.currentUser,
        self.current_week
    )

    # Places the activities based on which day they occurred on
    count = 1
    for a in activities:
        # Creates an ActivityBox frame in the correct Day frame
        box = ActivityBoxObject.ActivityBox(
            self.weekday(a.id),
            a
        )
        box.grid(column=0, row=count, ipadx=3, ipady=3, padx=3, pady=3)
        count += 1
```

Fig. 20 - refresh method of HomeScreen class

```
def weekday(self, stamp):
    # Returns the Day frame the activity should be placed in
    # Parses time stamp
    day_parts = stamp.split()[0].split('-')
    day = calendar.weekday(
        int(day_parts[0]),
        int(day_parts[1]),
        int(day_parts[2]))
    frame = self.days[day]
    return frame
```

Fig. 21 - weekday method of HomeScreen class used in the refresh method, returns

```python
class ActivityBox(tk.Frame):

    def __init__(self, parent, a):

        # Initializes frame
        tk.Frame.__init__(self, parent)
        self.configure(background='white')

        # Fonts used in frame
        self.fonts = {
            'ActivityTitle': ('Roboto', '20', 'bold'),
            'Username': ('Roboto', '14', 'bold'),
            'Labels': ('Roboto', '12'),
            'EditButton': ('Roboto', '8')
        }

        # Compiles all necessary information
        self.activity = a
        self.sport = a.sport
        self.title = a.title
        self.time = float(a.time)
        self.distance = float(a.distance)
        self.description = a.description
        self.id = a.id
        self.name = SheetTools.get_name(KTrackerApp.currentUser)

        # For skis, distance in km, pace in min/km
        if self.sport == 'Ski':
            self.distance = self.distance * 1.60934
            self.pace = self.time / self.distance
            seconds = int((self.pace % 1) * 60)
            minutes = int(self.pace - (self.pace % 1))
            self.pace = str(minutes) + ':' + str(seconds) + " /km"
            self.distance = str(round(self.distance, 2)) + " km"

        # For runs, distance in mi, pace in min/mi
        elif self.sport == 'Run':
            self.pace = self.time / self.distance
            seconds = int((self.pace % 1) * 60)
            minutes = int(self.pace)
            self.pace = str(minutes) + ':' + str(seconds) + " /mi"
            self.distance = str(round(self.distance, 2)) + " mi"

        # for rides, distance in mi, pace in mph
        elif self.sport == 'Bike':
            self.pace = self.distance / (self.time / 60)
            self.pace = str(round(self.pace, 2)) + " mph"
            self.distance = str(round(self.distance, 2)) + " mi"
```

Fig. 22 - ActivityBox class, formats activity data for presentation on HomeScreen

```
        # Parses time into h:m or m:s
        minutes = self.time
        if minutes >= 60:
            hours = minutes // 60
            minutes -= hours * 60
            self.time = str(int(hours)) + 'h ' + str(int(minutes)) + 'm'
        else:
            seconds = int((minutes % 1) * 60)
            self.time = str(int(minutes)) + 'm ' + str(seconds) + 's'

        # Initialize Widgets
        time_text = tk.Label(self, text=self.time, bg='white')
        distance_text = tk.Label(self, text=self.distance, bg='white')
        description_text = tk.Label(self, text=self.description, bg='white')
        pace_text = tk.Label(self, text=self.pace, bg='white')
        title_text = tk.Label(self, text=self.title, font=self.fonts['ActivityTitle'], bg='white')

        distance_label = tk.Label(self, text='Distance', font=self.fonts['Labels'], bg='white')
        pace_label = tk.Label(self, text='Pace', font=self.fonts['Labels'], bg='white')
        time_label = tk.Label(self, text='Time', font=self.fonts['Labels'], bg='white')

        delete_button = ttk.Button(
            self,
            text='Edit Activity',
            command=lambda: EditActivityWindow.EditActivityWindow(self.activity)
        )

        # Place Widgets
        time_label.grid(column=2, row=3, sticky='w')
        distance_label.grid(column=0, row=3, sticky='w')
        pace_label.grid(column=1, row=3, sticky='w')
        title_text.grid(column=0, columnspan=2, row=1, sticky='w')

        description_text.grid(column=0, columnspan=3, row=2, sticky='w')
        distance_text.grid(column=0, row=4, sticky='w')
        pace_text.grid(column=1, row=4, sticky='w')
        time_text.grid(column=2, row=4, sticky='w')

        delete_button.grid(column=0, columnspan=3, row=5, pady=3)
```

Fig. 23 - ActivityBox class (continued)

ProfileScreen

Displays a user's statistics, allows them to change their password.

```
import tkinter as tk
from tkinter import ttk

import ChangePasswordWindow
import KTrackerApp
import StatisticsTools
```

Fig. 24 - ProfileScreen.py imports

```
class ProfileScreen(tk.Frame):
    def __init__(self, parent, controller):

        tk.Frame.__init__(self, parent)
        self.controller = controller
        self.fonts = {
            'TitleText': ('Roboto', '25', 'bold'),
            'SportTitles': ('Roboto', '20', 'bold'),
            'Stats': ('Roboto', '15')
        }
        self.name = 'Profile'

        # Gets a Statistics object for the current user
        self.stats = StatisticsTools.generate_stats(KTrackerApp.currentUser)

        # Place username and change password button
        self.username_label = tk.Label(
            self,
            text=('Username : ' + KTrackerApp.currentUser)
        )

        self.change_pass = ttk.Button(
            self,
            text='Change Password',
            command=ChangePasswordWindow.ChangePasswordWindow
        )

        self.username_label.grid(column=0, row=0)

        self.change_pass.grid(column=0, row=1)

        # Initialize Labels
        self.stats_label = tk.Label(self, text='Statistics', font=self.fonts['TitleText'])
        self.totals_label = tk.Label(self, text='All Activities', font=self.fonts['SportTitles'])
        self.ski_label = tk.Label(self, text='Skis', font=self.fonts['SportTitles'])
        self.run_label = tk.Label(self, text='Runs', font=self.fonts['SportTitles'])
        self.bike_label = tk.Label(self, text='Rides', font=self.fonts['SportTitles'])
```

Fig. 25 - Profile Screen, displays total and average times/distance for all sports

```
        # Totals Labels
        self.totals_label = tk.Label(
            self,
            text='All Activities',
            font=self.fonts['SportTitles']
        )
        self.totals_count_label = tk.Label(
            self,
            text='Count',
            font=self.fonts['Stats']
        )
        self.totals_distance_label = tk.Label(
            self,
            text='Distance (mi)',
            font=self.fonts['Stats']
        )
        self.totals_time_label = tk.Label(
            self,
            text='Time',
            font=self.fonts['Stats']
        )

        # Averages Labels
        self.averages_label = tk.Label(
            self,
            text='Averages', font=self.fonts['Stats']
        )
        self.averages_distance_label = tk.Label(
            self,
            text='Distance (mi)', font=self.fonts['Stats']
        )
        self.averages_time_label = tk.Label(
            self,
            text='Time', font=self.fonts['Stats']
        )
```

Fig. 26 - ProfileScreen (continued)

```
        # Initialize Data
        # All activities Labels
        self.total_count = tk.Label(
            self,
            text=str(self.stats.total.total_count),
            font=self.fonts['Stats']
        )
        self.total_distance = tk.Label(
            self,
            text=str(self.stats.total.total_distance),
            font=self.fonts['Stats']
        )
        self.total_time = tk.Label(
            self,
            text=str(self.stats.total.total_time),
            font=self.fonts['Stats']
        )
        self.total_average_distance = tk.Label(
            self,
            text=str(round(self.stats.total.average_distance, 2)),
            font=self.fonts['Stats']
        )
        self.total_average_time = tk.Label(
            self,
            text=str(round(self.stats.total.average_time, 2)),
            font=self.fonts['Stats']
        )

        # Skis Labels
        self.ski_count = tk.Label(
            self,
            text=str(self.stats.ski.total_count),
            font=self.fonts['Stats']
        )
        self.ski_distance = tk.Label(
            self,
            text=str(self.stats.ski.total_distance),
            font=self.fonts['Stats']
        )
        self.ski_time = tk.Label(
            self,
            text=str(self.stats.ski.total_time),
            font=self.fonts['Stats']
        )
        self.ski_average_distance = tk.Label(
            self,
            text=str(round(self.stats.ski.average_distance, 2)),
            font=self.fonts['Stats']
        )
        self.ski_average_time = tk.Label(
            self,
            text=str(round(self.stats.ski.average_time, 2)),
            font=self.fonts['Stats']
        )
```

Fig. 27 - ProfileScreen (continued)

```
        # Runs Labels
        self.run_count = tk.Label(
            self,
            text=str(self.stats.run.total_count),
            font=self.fonts['Stats']
        )
        self.run_distance = tk.Label(
            self,
            text=str(self.stats.run.total_distance),
            font=self.fonts['Stats']
        )
        self.run_time = tk.Label(
            self,
            text=str(self.stats.run.total_time),
            font=self.fonts['Stats']
        )
        self.run_average_distance = tk.Label(
            self,
            text=str(round(self.stats.run.average_distance, 2)),
            font=self.fonts['Stats']
        )
        self.run_average_time = tk.Label(
            self,
            text=str(round(self.stats.run.average_time, 2)),
            font=self.fonts['Stats']
        )

        # Bike rides Labels
        self.bike_count = tk.Label(
            self,
            text=str(self.stats.bike.total_count),
            font=self.fonts['Stats']
        )
        self.bike_distance = tk.Label(
            self,
            text=str(self.stats.bike.total_distance),
            font=self.fonts['Stats']
        )
        self.bike_time = tk.Label(
            self,
            text=str(self.stats.bike.total_time),
            font=self.fonts['Stats']
        )
        self.bike_average_distance = tk.Label(
            self,
            text=str(round(self.stats.bike.average_distance, 2)),
            font=self.fonts['Stats']
        )
        self.bike_average_time = tk.Label(
            self,
            text=str(round(self.stats.bike.average_time, 2)),
            font=self.fonts['Stats']
        )
```

Fig. 28 - ProfileScreen (continued)

```
        # Place Labels
        self.stats_label.grid(column=1, columnspan=2, row=2, pady=15, sticky='w')
        self.averages_label.grid(column=5, row=3, sticky='w', padx=10)
        self.totals_label.grid(column=1, row=5, sticky='w')
        self.ski_label.grid(column=1, row=6, sticky='w')
        self.run_label.grid(column=1, row=7, sticky='w')
        self.bike_label.grid(column=1, row=8, sticky='w')
        self.totals_count_label.grid(column=2, row=4, sticky='w', padx=10)
        self.totals_distance_label.grid(column=3, row=4, sticky='w', padx=10)
        self.totals_time_label.grid(column=4, row=4, sticky='w', padx=10)
        self.averages_distance_label.grid(column=5, row=4, sticky='w', padx=10)
        self.averages_time_label.grid(column=6, row=4, sticky='w', padx=10)

        # All activities
        self.total_count.grid(column=2, row=5)
        self.total_distance.grid(column=3, row=5)
        self.total_time.grid(column=4, row=5)
        self.total_average_distance.grid(column=5, row=5)
        self.total_average_time.grid(column=6, row=5)

        # Skis
        self.ski_count.grid(column=2, row=6)
        self.ski_distance.grid(column=3, row=6)
        self.ski_time.grid(column=4, row=6)
        self.ski_average_distance.grid(column=5, row=6)
        self.ski_average_time.grid(column=6, row=6)

        # Runs
        self.run_count.grid(column=2, row=7)
        self.run_distance.grid(column=3, row=7)
        self.run_time.grid(column=4, row=7)
        self.run_average_distance.grid(column=5, row=7)
        self.run_average_time.grid(column=6, row=7)

        # Bike rides
        self.bike_count.grid(column=2, row=8)
        self.bike_distance.grid(column=3, row=8)
        self.bike_time.grid(column=4, row=8)
        self.bike_average_distance.grid(column=5, row=8)
        self.bike_average_time.grid(column=6, row=8)
```

Fig. 29 - ProfileScreen (continued)

AddActivityScreen

Allows users to add new activities, specifying sport, time, distance, ect.

```
import datetime
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk

import ActivityDataObject
```

Fig. 30 - AddActivityScreen.py imports

```
class AddActivityScreen(tk.Frame):

    def __init__(self, parent, controller):

        # Creates and configures frame
        tk.Frame.__init__(self, parent)
        self.controller = controller
        self.name = 'Add Activity'

        # Values for user selections
        self.sports = ['Ski', 'Run', 'Bike']
        self.units = ['miles', 'yards', 'kilometers', 'meters']
        self.months = [
            'January', 'February', 'March', 'April',
            'May', 'June', 'July', 'August',
            'September', 'October', 'November', 'December'
        ]
        self.days = [a for a in range(1, 32)]
        self.years = [a for a in range(current_year(), 1970, -1)]

        # Initialize Widgets
        self.distance_label = tk.Label(self, text='Distance')
        self.duration_label = tk.Label(self, text='Duration')
        self.sport_label = tk.Label(self, text='Sport')
        self.title_label = tk.Label(self, text='Title')
        self.description_label = tk.Label(self, text='Description')
        self.date_label = tk.Label(self, text='Date')

        self.distance_input = tk.Entry(self)
        self.duration_input = tk.Entry(self)
        self.unit_input = ttk.Combobox(self, values=self.units, state='readonly', width=10)
        self.unit_input.set(self.units[0])
        self.sport_input = ttk.Combobox(self, values=self.sports, state="readonly", width=10)
        self.sport_input.set(self.sports[0])
        self.title_input = tk.Entry(self)
        self.description_input = tk.Text(self, height=5, width=40, wrap='word')
        self.month_input = ttk.Combobox(self, values=self.months, state='readonly', width=10)
        self.month_input.set(self.current_month())
        self.day_input = ttk.Combobox(self, values=self.days, width=3)
        self.day_input.set(current_day())
        self.year_input = ttk.Combobox(self, values=self.years, width=5)
        self.year_input.set(current_year())
```

Fig. 31 - AddActivityScreen, inputs for sport, time, ect.

```
self.create = ttk.Button(
    self,
    text='Create',
    command=lambda:
        self.add_activity(
            [
                self.sport_input.get(),
                self.title_input.get(),
                self.duration_input.get(),
                self.distance_input.get(),
                self.description_input.get('1.0', tk.END)[:-1],
                self.day_input.get(),
                self.month_input.get(),
                self.year_input.get(),
                self.unit_input.get()
            ]
        )
)
self.cancel = ttk.Button(self, text='Cancel', command=self.clear)

# Place Widgets
self.distance_label.grid(column=0, row=0, sticky='w')
self.duration_label.grid(column=2, columnspan=2, row=0, sticky='w')
self.sport_label.grid(column=4, row=0, sticky='w')
self.title_label.grid(column=0, row=2, sticky='w')
self.description_label.grid(column=0, row=4, sticky='w')

self.distance_input.grid(column=0, row=1, sticky='w')
self.unit_input.grid(column=1, row=1, sticky='w')
self.duration_input.grid(column=2, columnspan=2, row=1, sticky='w')
self.sport_input.grid(column=4, row=1, sticky='w')
self.title_input.grid(column=0, row=3, sticky='w')
self.month_input.grid(column=1, row=3, sticky='w')
self.day_input.grid(column=2, row=3, sticky='w')
self.year_input.grid(column=3, row=3, sticky='w')
self.description_input.grid(column=0, columnspan=3, row=5, sticky='w')

self.create.grid(column=0, row=6)
self.cancel.grid(column=1, row=6)
```

Fig. 32 - AddActivityScreen (continued)

```
def clear(self):
    # Resets all inputs after an activity is submitted
    self.distance_input.delete(0, len(self.distance_input.get()))
    self.duration_input.delete(0, len(self.duration_input.get()))
    self.sport_input.delete(0, len(self.sport_input.get()))
    self.sport_input.set(self.sports[0])
    self.title_input.delete(0, len(self.title_input.get()))
    self.description_input.delete('1.0', tk.END)
    self.unit_input.set(self.units[0])
    self.year_input.set(current_year())
    self.month_input.set(self.current_month())
    self.day_input.set(current_day())
```

Fig. 33 - clear method of AddActivityScreen, called after activities are added, clears all inputs

```
# All parse current timestamp and return desired part
def current_day():
    today = str(datetime.datetime.today())
    today = int(today.split()[0].split('-')[2])

    return today


def current_month(self):
    today = str(datetime.datetime.today())
    today = int(today.split()[0].split('-')[1])
    month = self.months[today-1]

    return month


def current_year():
    today = str(datetime.datetime.today())
    today = int(today.split()[0].split('-')[0])

    return today
```

Fig. 34 - current_ methods of AddActivityScreen, returns parts of the current date

ChangePasswordWindow

Accessed through ProfileScreen, allows users to set a new password.

```
import tkinter as tk
from tkinter import messagebox

import SheetTools
import KTrackerApp
```

Fig. 35 - ChangePasswordWindow.py imports

```
class ChangePasswordWindow:

    def __init__(self):

        # Create and name window
        self.Frame = tk.Tk()
        tk.Tk.wm_title(
            self.Frame,
            string='KTracker | Change Password'
        )

        # Fonts used in window
        self.fonts = {
            'TitleText': ('Roboto', '25', 'bold')
        }

        # Initialize Widgets
        self.change_pass_label = tk.Label(self.Frame, text='Change Password', font=self.fonts['TitleText'])
        self.old_pass_label = tk.Label(self.Frame, text='Old Password')
        self.new_pass1_label = tk.Label(self.Frame, text='New Password')
        self.new_pass2_label = tk.Label(self.Frame, text='Confirm')

        self.old_pass_input = tk.Entry(self.Frame, show='\u2022')
        self.new_pass1_input = tk.Entry(self.Frame, show='\u2022')
        self.new_pass2_input = tk.Entry(self.Frame, show='\u2022')

        self.submit = tk.Button(self.Frame, text='Submit', command=self.final)

        # Place Widgets
        self.old_pass_input.grid(column=1, row=1)
        self.new_pass1_input.grid(column=1, row=2)
        self.new_pass2_input.grid(column=1, row=3)

        self.change_pass_label.grid(column=0, columnspan=2, row=0)
        self.old_pass_label.grid(column=0, row=1)
        self.new_pass1_label.grid(column=0, row=2)
        self.new_pass2_label.grid(column=0, row=3)

        self.submit.grid(column=1, row=4)
```

Fig. 36 - ChangePasswordWindow class, inputs for old password, and verifying new password

```python
    def change_pass(self):
        # Retrieve password inputs
        old_pass = self.old_pass_input.get()
        new_pass1 = self.new_pass1_input.get()
        new_pass2 = self.new_pass2_input.get()

        # Check if the old password is correct
        if not SheetTools.get_password(KTrackerApp.currentUser) == old_pass:
            tk.messagebox.showerror('Error', 'Current password is incorrect.')
            return 1

        # If the new passwords match, update the document and close the window
        if new_pass1 == new_pass2:
            sheet = SheetTools.get_worksheet(KTrackerApp.currentUser)
            sheet.update_acell('A1', new_pass1)
            tk.messagebox.showinfo('Done', 'Password changed successfully')
            self.Frame.destroy()
        else:
            tk.messagebox.showerror('Error', 'Passwords do not match.')
            return 1
```

Fig. 37 - change_pass method of ChangePasswordWindow class

EditActivtyWindow

Allows users to edit an activity, autofills inputs with old values of the activity.

```
import datetime
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk

import ActivityDataObject
```

Fig. 38 - EditActivityWindow.py imports

```
class EditActivityWindow:

    def __init__(self, a):

        self.Frame = tk.Tk()
        tk.Tk.wm_title(
            self.Frame,
            string='KTracker | Edit Activity'
        )
        self.Frame.geometry('500x250')
        self.activity = a

        # Values for user inputs
        self.sports = ['Ski', 'Run', 'Bike']
        self.units = ['miles', 'yards', 'kilometers', 'meters']
        self.months = [
            'January', 'February', 'March', 'April',
            'May', 'June', 'July', 'August',
            'September', 'October', 'November', 'December'
        ]
        self.days = [a for a in range(1, 32)]
        self.years = [a for a in range(current_year(), 1970, -1)]

        # Initializes Widgets
        self.distance_label = tk.Label(self.Frame, text='Distance')
        self.duration_label = tk.Label(self.Frame, text='Duration')
        self.sport_label = tk.Label(self.Frame, text='Sport')
        self.title_label = tk.Label(self.Frame, text='Title')
        self.description_label = tk.Label(self.Frame, text='Description')
        self.date_label = tk.Label(self.Frame, text='Date')

        self.distance_input = tk.Entry(self.Frame)
        self.distance_input.insert('0', self.activity.distance)
        self.duration_input = tk.Entry(self.Frame)
        self.duration_input.insert('0', self.activity.time)
        self.title_input = tk.Entry(self.Frame)
        self.title_input.insert('0', self.activity.title)
        self.description_input = tk.Text(self.Frame, height=5, width=40, wrap='word')
        self.description_input.insert('1.0', self.activity.description)

        self.unit_input = ttk.Combobox(self.Frame, values=self.units, state='readonly', width=10)
        self.unit_input.set(self.units[0])
        self.sport_input = ttk.Combobox(self.Frame, values=self.sports, state="readonly", width=10)
        self.sport_input.set(self.activity.sport)
        self.month_input = ttk.Combobox(self.Frame, values=self.months, state='readonly', width=10)
        self.month_input.set(self.month(self.activity.id))
        self.day_input = ttk.Combobox(self.Frame, values=self.days, width=3)
        self.day_input.set(day(self.activity.id))
        self.year_input = ttk.Combobox(self.Frame, values=self.years, width=5)
        self.year_input.set(year(self.activity.id))
```

Fig. 39 - EditActivityWindow, inputs for sport, time, distance, ect.

```
        self.create = ttk.Button(
            self.Frame,
            text='Create',
            command=lambda:
            self.edit_activity(
                [
                    self.sport_input.get(),
                    self.title_input.get(),
                    self.duration_input.get(),
                    self.distance_input.get(),
                    self.description_input.get('1.0', tk.END)[:-1],
                    self.day_input.get(),
                    self.month_input.get(),
                    self.year_input.get(),
                    self.unit_input.get(),
                    self.activity
                ]
            )
        )
        self.cancel = ttk.Button(self.Frame, text='Cancel', command=lambda: self.Frame.destroy())

        # Place Widgets
        self.distance_label.grid(column=0, row=0, sticky='w')
        self.duration_label.grid(column=2, columnspan=2, row=0, sticky='w')
        self.sport_label.grid(column=4, row=0, sticky='w')
        self.title_label.grid(column=0, row=2, sticky='w')
        self.description_label.grid(column=0, row=4, sticky='w')

        self.distance_input.grid(column=0, row=1, sticky='w')
        self.unit_input.grid(column=1, row=1, sticky='w')
        self.duration_input.grid(column=2, columnspan=2, row=1, sticky='w')
        self.sport_input.grid(column=4, row=1, sticky='w')
        self.title_input.grid(column=0, row=3, sticky='w')
        self.month_input.grid(column=1, row=3, sticky='w')
        self.day_input.grid(column=2, row=3, sticky='w')
        self.year_input.grid(column=3, row=3, sticky='w')
        self.description_input.grid(column=0, columnspan=3, row=5, sticky='w')

        self.create.grid(column=0, row=6)
        self.cancel.grid(column=1, row=6)
```

Fig. 40 - EditActivityWindow (continued)

```python
def edit_activity(self, a):

    old_activity = a[-1]

    # Verifies day - month pairs
    month_days = {
        'January': ['01', 31],
        'February': ['02', 28],
        'March': ['03', 31],
        'April': ['04', 30],
        'May': ['05', 31],
        'June': ['06', 30],
        'July': ['07', 31],
        'August': ['08', 31],
        'September': ['09', 30],
        'October': ['10', 31],
        'November': ['11', 30],
        'December': ['12', 31]
    }
    if not int(a[5]) <= month_days[a[6]][1]:
        messagebox.showerror(
            'Error',
            'Invalid day : ' + a[5] + ' for month : ' + a[6]
        )
        return 1

    # Parses date into valid timestamp
    a[5] = str(a[7]) + '-' + str(month_days[a[6]][0]) + '-' + str(a[5])

    # Ensures that time is float
    try:
        a[2] = round(float(a[2]), 2)
    except ValueError:
        self.duration_input.delete(
            0,
            len(self.duration_input.get())
        )
        messagebox.showerror(
            'Error',
            'Invalid time value'
        )
        return 1
```

Fig. 41 - edit_activty method, checks new values, then calls SheetTools.edit_activity to edit the entry in the database

```
        # Ensures that distance is float
        try:
            a[3] = round(float(a[3]), 2)
        except ValueError:
            self.distance_input.delete(
                0,
                len(self.distance_input.get())
            )
            messagebox.showerror(
                'Error',
                'Invalid distance value'
            )
            return 1

        # Ensures that a title was added
        if not a[1]:
            messagebox.showerror(
                'Error',
                'Please input a title'
            )
            return 1

        # Converts all units to miles
        if a[8] == 'yards':
            a[3] = round(a[3] / 1760, 2)
        if a[8] == 'kilometers':
            a[3] = round(a[3] / 1.60934, 2)
        if a[8] == 'meters':
            a[3] = round(a[3] / 1609.34, 2)

        # Converts activity to ActivityData object
        a = ActivityDataObject.ActivityData(a)

        # Edits the database record of the activity and closes the window
        a.edit(old_activity)

        self.Frame.destroy()
```

Fig. 42 - edit_activity (continued)

Data Interactions

All of the following methods are stored in the SheetTools.py file, and imported into other files as needed. They all are for the most part low-level interactions with the database that occur often.

```
import gspread
from oauth2client.service_account import ServiceAccountCredentials

import ActivityDataObject
```

Fig. 43 - SheetTools.py imports

```
def open_database():
    scope = [
        'https://spreadsheets.google.com/feeds',
        'https://www.googleapis.com/auth/drive'
    ]
    credentials = ServiceAccountCredentials.from_json_keyfile_name(
        'client_secret.json',
        scope
    )  # Retrieves credentials from client_secret.json file
    gc = gspread.authorize(credentials)  # Authorizes GSpread
    database = gc.open("KTrackerDatabase")  # Opens KTrackerDatabase sheet
    return database
```

Fig. 44 - Opening the database spreadsheet

```
def check_user(user):
    sheet = open_database()  # Retrieves the database

    worksheets = sheet.worksheets()  # Gets a list of all users

    if user in worksheets:  # Checks if the user is in the list
        return True
    else:
        return False
```

Fig. 45 - Check if a user exists

```
def get_user_data(user):
    sheet = open_database()  # Retrieves the database

    try:
        worksheet = sheet.worksheet(user)  # Gets the user's worksheet
        data = worksheet.get_all_values()  # Parses into a 2D array
        return data
    except ValueError:
        return 1
```

Fig. 46 - Getting all of user's data

```
def get_password(user):
    data = get_user_data(user)  # Retrieves a user's worksheet as a 2D array

    password = data[0][0]  # Gets the password from the array

    return password
```

Fig. 47 - Get a user's password

```
def get_stats(user):
    data = get_user_data(user)  # Retrieves a user's worksheet as a 2D array

    stats = [[] for _ in range(4)]  # Generates a 2D array for storing statistics

    for i in range(4, 8):
        for j in range(1, 6):
            stats[i-4].append(data[i][j])  # Adds the statistics to the array

    return stats
```

Fig. 48 - Get a user's statistics

```
def get_activities(user):
    data = get_user_data(user)  # Retrieves a user's worksheet as a 2D array

    num_activities = len(data) - 11  # Int value of number of activities for the user

    activities = [[] for _ in range(num_activities)]  # Generates 2D array for activities

    for i in range(11, 11+num_activities):  # Copies activities from worksheet
        for j in range(0, 6):
            activities[i-11].append(data[i][j])

    for i in range(len(activities)):  # Converts all the activities into Activity objects
        activities[i] = ActivityObject.Activity(activities[i])

    return activities
```

Fig. 49 - Get all a user's activities as Activity objects

```
def add_activity(user, a):
    database = open_database()
    data = database.worksheet(user)  # Retrieves a user's worksheet

    data.insert_row(a, 12)  # Adds the activity as a new row on the sheet
```

Fig. 50 - Add a new activity

```python
def delete_activity(user, activity_id):
    database = open_database()
    sheet = database.worksheet(user)  # Retrieves a user's worksheet
    data = sheet.get_all_values()  # Parses as 2D array

    num_activities = len(data) - 11  # Int value of number of activities for the user

    for i in range(11, 11+num_activities):  # Searches for the matching timestamp
        if int(data[i][5]) == activity_id:
            sheet.delete_row(i+1)  # Deletes the matching activity
            break
```

Fig. 51 - Delete an activity by timestamp

```python
def add_user(username, password1, password):
    try:  # If the user exists, does nothing
        get_user_data(username)
    except ValueError:
        if password == password1:  # Checks if the passwords match
            database = open_database()
            database.add_worksheet(
                title=username,
                rows='100',
                cols='10'
            )  # Adds a worksheet for the new user
            worksheet = database.worksheet(username)  # Retrieves the new worksheet
            data = template  # copies the saved template for users
            data[0][0] = password  # changes the password cell
            for i in range(len(data)-1, -1, -1):
                worksheet.insert_row(data[i], index=1)  # Adds the template to the sheet
```

Fig. 52 - Add new user

```python
def get_worksheet(user):
    database = open_database()  # Opens the database
    worksheet = database.worksheet(user)  # Retrieves the user's worksheet
    return worksheet
```

Fig. 53 - Get a user’s worksheet

```python
def edit_activity(user, new, old):
    data = get_user_data(user)  # Retrieves a user's worksheet as a 2D array

    num_activities = len(data) - 11  # Int value of number of activities for the user

    for i in range(11, 11+num_activities):  # Checks if the activity remained the same
        same = True
        for j in range(0, 6):
            if not data[i][j] == old[j]:  # Compares activity to all saved in Database
                same = False

        if same:  # If the activity had been edited
            data = get_worksheet(user)
            data.delete_row(i+1)  # Removes the old activity
            add_activity(user, new)  # Adds the edited version
            break
```

Fig. 54 - Edit Activity

```
import KTrackerApp
import SheetTools
```

Fig. 55 - ActivityDataObject.py imports

```
class ActivityData:

    def __init__(self, data):
        # Parses data from array into object
        self.sport = data[0]
        self.title = data[1]
        self.time = data[2]
        self.distance = data[3]
        self.description = data[4]
        self.id = data[5]

        # Array off all information
        self.attributes = [
            self.sport,
            self.title,
            self.time,
            self.distance,
            self.description,
            self.id
        ]
```

Fig. 56 - ActivityData class used for manipulating activities

```
def add(self):
    # Parses data back into array
    data = [0 for _ in range(6)]
    for a in range(len(self.attributes)):
        data[a] = self.attributes[a]

    # Adds the activity to the database
    SheetTools.add_activity(
        KTrackerApp.currentUser,
        data
    )
```

Fig. 57 - add method of ActivityData class

```
def edit(self, old_activity):
    # Parses new and old data
    new = [0 for _ in range(6)]
    old = [0 for _ in range(6)]
    for a in range(len(self.attributes)):
        new[a] = self.attributes[a]
        old[a] = old_activity.attributes[a]

    # Makes all necessary edits
    SheetTools.edit_activity(
        KTrackerApp.currentUser,
        new,
        old
    )
```

Fig. 58- edit method of ActivtyData Class