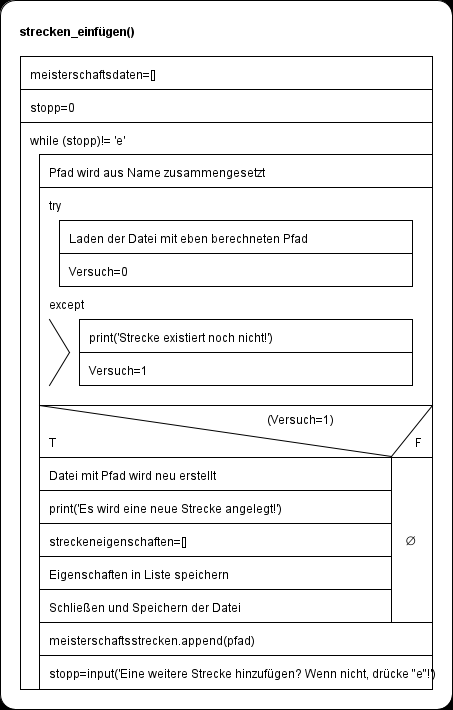
**Dokumentation zu Mebe V1.0.0**

**11. Septmeber 2022**

Willkommen bei Mebe! Geöffnet in der Entwicklungsumgebung erschlägt einem das Projekt sofort und scheint bei einer Ausführung des Programms auch keine richtigen Aufschlüsse zu zeigen… Wofür ist das Programm überhaupt da?  
Mebe hat einen ungewöhnlichen Aufgabenbereich, den wohl noch kein Programm vorher hatte. Es soll Motorsportmeisterschaften berechnen. Und nicht nur irgendwelche, sondern die, die man sich selbst ausgedacht hat!  
Mebe ist als Hilfe für mich schon seit zwei Jahren in Planung, damals sollte es ein Freund für mich programmieren. Da er aber sein Abitur geschrieben hatte, verlief das Projekt erstmal im Sand, bis es in einer Excel-Datei landete, anhand der ich ein konkretes Rennen entwickelt habe. Diese Tabelle diente auch als Vorlage für Mebe, die fast eins zu eins im „Berechnungsmodus“ wiederzufinden ist. Diese Projektarbeit hat mir den Antrieb gegeben, ein Projekt zu erschaffen, von dem ich schon ewig träumte!  
Mebe wurde mit hohen Anforderungen konfrontiert. Die klassische „Blitzländische Tourenwagen Meisterschaft“ sollte nicht nur vollständig berechnet werden, jedes Rennergebnis und jedes Meisterschaftsergebnis sollte auch gespeichert und (später) formatiert werden! Ein großes Problem stellte dar, dass die BTM jedes Jahr anders aufgebaut war, etwa durch andere Fahrzeuge, ein anderer Rennkalender oder andere Fahrer. Mebe ermöglicht es nun, die Rangfolge völlig individueller Motorsportmeisterschaften zu berechnen.  
Startet man das Programm, landet man in einer Menüführung. Die jeweiligen Zahlen bieten die Auswahlmöglichkeiten an, welche zur Option stehen. Die „3“ dürfte für alle Einsteiger das wichtigste sein. Mit diesem Menü kann man seine eigene Meisterschaft erstellen. Nachdem der Name erfragt wurde, müssen Strecken hinzugefügt werden. Ist die Strecke in der Datenbank noch nicht vorhanden, so wird eine neue Strecke angelegt, deren Eigenschaften definiert werden. Später gilt selbiges bei den Fahrern. Die Abfrage nach den Fahrzeugen dient der Absicherung, dass alle nötigen Dateien gespeichert sind. In einem weiteren Update soll es hinfällig werden.  
Diese Möglichkeit der Wiederverwendung von Dateien ist eine sehr erleichternde Sache, da sich der Datensatz der unterschiedlichen Meisterschaften nur wenig voneinander unterscheidet. Ein einziges Problem gibt es nur, dass für jeden Meisterschaftstyp ein neues Fahrerprofil angelegt werden muss, wenn er mit einem völlig abweichenden Fahrzeug fährt. Dies soll über längeren Zeitraum noch verändert werden, sodass das Programm automatisch erkennt, welches der vielen Fahrzeuge der Fahrer genau in dieser Meisterschaft fährt.   
Ein weiterer Dorn im Auge ist noch die umständliche Bedienung, die aufgrund der Zeit noch nicht in einem Extrafenster ablaufen kann und es erheblich einfacher machen würde. Fehler passieren leider leicht. So gibt es auch, im Hauptmenü wieder angelangt, die Möglichkeit, die Meisterschaft zu bearbeiten. Wählt man also die „4“ aus, so wird man in ein Untermenü geführt, wo gebeten wird, einen Buchstaben einzutippen. Es besteht die Möglichkeit, Streckeneigenschaften, Fahrereigenschaften und Fahrzeugeigenschaften zu bearbeiten, was aufgrund eines Fehlers noch mit Neudefinitionen realisiert wird.. Über dieses Menü ist es auch möglich, neue Profile anzulegen, ohne dass sie in einer Meisterschaft direkt verwendet werden müssen.  
Hat man sich durch die Neudefinition durchgequält, landet man wieder im Menü. Mit der „5“ kann das Programm beendet werden, stattdessen kann man aber auch die „2“ drücken, die später die Funktion ausführen soll, schon berechnete Meisterschaften wieder anzuschauen.  
Zuvor sollte man aber schon eine Meisterschaft berechnet haben, das Hauptfeature des Programms.  
Hierfür wird in einer Schleife jede Strecke einzeln berechnet. In jeweiligen Dateien, die das Programm vorher angelegt hatte, sind alle relevanten Informationen enthalten, um für jeden Fahrer ein eigenes Rennergebnis zu berechnen. Viel hängt vom Zufall ab, so sind diese beispielsweise ausschlaggebend dafür, ob der Fahrer Extrapunkte bekommt oder ob er eben einen Unfall baut. Andererseits ist viel vom Können oder vom Gemüt abhängig, so steigt bei einem sehr aggressiven Fahrer beispielsweise das Risiko eines Unfalls, besonders bei nasser Strecke. Oder hat der Fahrer eine Vorliebe für schnelle Strecken und der befahrene Rundkurs ist sehr schnell, so bekommt er Extrapunkte für seine Bewertung. Hauptindikator ist aber das vorher eingegebene Grundkönnen. Diese Extrapunkte geben quasi an, wie gut sich ein Fahrer in diesem Rennen geschlagen hat. Dieses Ergebnis wird gemeinsam mit dem passenden Namen in einer Ergebnisliste zwischengespeichert und dann nach dieser Punktzahl sortiert (sind Punkte gleich, so entscheidet das Grundkönnen). Im nächsten Schritt werden diese Punkte von der richtigen Punktzahl nach Definition überschrieben, zum Beispiel erhält der Erstplatzierte 20 Punkte, der Zweite 18 usw. Später soll es auch möglich sein, jeder Meisterschaft ein eigenes Punktesystem zuzuordnen, was aber für das Erste nicht wichtig war und deshalb noch nicht umgesetzt wurde.   
Doch nicht nur die Ergebnisliste des Rennens wird gespeichert, für jeden Fahrer wird eine temporäre Datei angelegt, in der die Zwischenergebnisse, also die errungenen Punkte, in einer Liste nach Pilot gespeichert sind. Denn die sind jetzt von Bedeutung, wenn es an die Gesamtplatzierung geht. Hierfür wird wie bei einer Streckenberechnung vorgegangen. Jeder Fahrer wird mit seiner temporären Datei aufgerufen und seine Punktzahl wird addiert. Anschließend wird der Fahrer mit seiner Punktzahl wieder in einer Liste gespeichert, wo auch andere Fahrer gespeichert werden. Zum Schluss wird die Rangfolge ermittelt, indem die Liste nach der Punktzahl sortiert wird. Diese Liste wird gespeichert und ausgegeben. Et voila! Das ist Mebes Hauptaufgabe!  
Mebe selbst ist sehr komplex, was es nicht einfach macht, ein konkretes Struktogramm zu erstellen. Allerdings habe ich ein paar Zeilen gefunden, die sich umsetzen ließen.



Abbruchbedingung bestimmen

Richtiger Name, den man angab, wird eingesetzt

Versucht, die Anweisungen auszuführen.  
Wird eine Fehlermeldung zurückgegeben,   
springt das Programm zu „except“.

Es wird versucht, die Datei mit obigem Pfad zu öffnen.

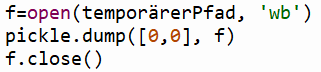
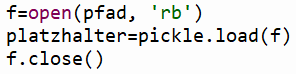
Kann er den try-Block nicht ausführen, sprich die Datei  
nicht laden, so muss es daran liegen, dass sie nicht   
existiert, da der Pfad nur den Dateinamen falsch setzen   
kann.

Datei wird neu erstellt, da sie nicht geladen werden konnte

Datei existiert schon, also muss sie nicht erstellt werden.

Datei wird in einer Liste vermerkt, auf welche die  
Berechnung später zugreifen wird und daraus liest, welche Dateien herangezogen werden müssen.

Abbruchvariable; wenn nicht auf „e“, wird weitere  
Strecke hinzugefügt.

Gezeigt ist der Ablauf, wie eine Strecke zu einer Meisterschaft hinzugefügt wird. Dieser ist mit dem Ablauf von Fahrern und Fahrzeugen nahezu identisch. Hervorheben möchte ich den try-Block, da er überprüft, ob die hinzuzufügende Datei schon vorhanden ist. Besonders bei Fahrern wird dadurch eine Menge Arbeit gespart, da man keine Eigenschaften mehr eintippen muss, weil sie schon gespeichert sind. Es ist zudem wichtig zu verstehen, dass nur die jeweiligen Pfade bei der Meisterschaft gespeichert werden. Dies erleichtert dem Programm um komplizierte Berechnungen, so muss man später im „Berechnungsmodus“ nur noch festlegen, wo die im Moment gesuchten Dateien zu finden sind, was durch die gegebene Struktur keine Schwierigkeit ist. Demnach wird zuerst in der Meisterschaftsdatei geschaut, diese geladen, dann gelesen, welche Strecken zu laden sind, dann wird eine Strecke berechnet, in dem jeder Fahrer aufgerufen wird und mit den Eigenschaften ein wenig jongliert wird. Nähere Erläuterungen sind als Stichpunkt neben dem Struktogramm vorzufinden.  
Mit Hilfe des Buches „Python 3 Schnelleinstieg“ von Michael Weigend (mitp-Verlag, 2021) habe ich viele neue Techniken gelernt, die mir sehr weitergeholfen haben. So unter anderem habe ich mir das Speichern und Laden von DAT-Dateien selbst beigebracht. Bei DAT-Dateien handelt es sich um Binärdateien, ich habe es als gespeicherte Liste kennengelernt. So kann man im Programm selbst eine Liste anfertigen, in der alle Daten eingetragen sind, die gespeichert werden sollen.

Speichern --> <-- Laden

Der Quelltext zum Speichern selbst ist ganz einfach. „f“ steht für einen sogenannten Stream, in der die Datei geöffnet ist. Existiert die Datei noch nicht, so wird eine neue erstellt, existiert sie schon, wird sie überschrieben (Bild links, speichern). Die Methode „open“ wird beim Laden und beim Speichern verwendet, es wird der Stream geöffnet. Der erste Parameter, „temporärerPfad“, im zweiten Bild „pfad“, ist in meinem Fall eine Variable. Sie steht für einen vorher durch das Programm festgelegten Dateipfad. Soll die Datei im selben Verzeichnis wie das Programm selbst stehen, so würde da nur der Name der Datei stehen, z. B. „Liste.dat“. „wb“ im zweiten Parameter gibt an, in welchen Modus der Stream geöffnet wird. „wb“ steht hier für „Write binar“, zumindest war es meine Merkhilfe. Steht stattdessen „rb“, „read binar“, dann wird die Datei nur zum Lesen freigegeben. Der Modus ist wichtig für die folgenden Handlungen, was mit der Datei geschehen soll. Beim Schreiben kann die Datei auch verändert werden, beim Lesen nicht.   
In der zweiten Zeile, die Methode pickle.dump() (es muss das Modul pickle importiert werden), wird dem Programm gesagt, dass ein Stream überschrieben werden soll, und zwar genau mit den Werten im ersten Parameter, in dem Fall eine Liste mit zwei Werten (0 und 0). Welcher Stream gemeint ist, gibt der zweite Parameter an. Die dritte Zeile, „f.close()“ ist wichtig, um den Stream zu schließen und die Datei zu speichern. Denn erst mit Abschließen des Streams wird die Datei überhaupt bearbeitet/erstellt. Vorher wurde es nur im Stream behandelt, eine Art Magnetband, der dann in die richtige Form gepackt wird.  
Eine Datei zu laden ist sehr ähnlich und unterscheidet sich nur in zwei Dingen. Der Modus der Methode „open“ (wie oben erklärt) und in der zweiten Zeile. Statt „pickle.dump()“ wird eine neue Methode verwendet, „pickle.load“, die nur einen Parameter hat, welcher Stream geladen werden soll (f). Diese Zeile liest quasi vor, was in der Datei gespeichert ist. Jetzt ist es wichtig, diese gelesenen Daten für das Programm verwendbar zu machen – man speichert sie in einer Variable, in dem Fall „platzhalter“. Der Inhalt der Binärdatei wird also eins zu eins auf die Variable übertragen. Das ist auch eigentlich das ganze Geheimnis hinter dem Speichern und Laden. Sie kommt im Programm sehr häufig vor, beispielsweise gleich in Zeile 41 bei der Streckendefinition (Bild rechts) oder in Zeile 419 bei der Punktberechnung der Fahrer am Ende der Meisterschaft (Bild links).  
Das Programm selbst zu schreiben war keine große Schwierigkeit. Am 24.7.2022 waren alle Vorüberlegungen abgeschlossen. Ich habe zu dieser Zeit die neu erlernten Methoden ausprobiert und bereits angefangen, das Programm zu schreiben. Schon ein Tag später waren die zwei Modi „Meisterschaft erstellen“ und „Meisterschaft bearbeiten“ fertig, woraufhin ich meine Zeit der Meisterschaftsberechnung widmete. Noch vor meinen Urlaub, am 29.7.2022, konnte ich auch diesen Modus fertig programmieren. Zurück in Dresden konnte ich am 11.8.2022 weiter arbeiten und am 14.8.2022 das Teilprogramm des Speicherns der Ergebnisse fertigstellen. Am Folgetag verkündete ich stolz, dass die eigentliche Programmierung fertiggestellt ist. Von nun an verbrachte ich die Zeit mit Debugging. Bis zum 30. August 2022 saß ich jeden zweiten Tag am Projekt, um die Fehler alle auszumerzen, bis der erste erfolgreiche Testdurchlauf geglückt ist. Die folgenden zwei Tage verbrachte ich damit, Formfehler zu korrigieren, ab 1. September 2022 gilt „Mebe V1.0.0“ als abgeschlossen. Weitere Updateversionen sind noch geplant, allerdings nicht mehr im Rahmen für diese Projektarbeit.  
Ich steckte sehr viel Zeit in das Programm, hauptsächlich in die Fehlerbehebung, die fast mehr Zeit in Anspruch genommen hatte, als die eigentliche Programmierung. Ich bin auch zufrieden mit meiner eigenen Zeiteinteilung, auch wenn ich die Abgabe bis auf den letzten Tag ausgereizt habe. Ich habe von Anfang an gewusst, dass ich mit hohen Anforderungen heran ging und war mir dessen bewusst, dass ich mehr Zeit investieren muss. Ich habe mich aber bewusst dafür entschieden, da es mir Freude bereitet hatte und ich Nutzen aus dem Projekt zog. Ich bin mit dem Ergebnis zufrieden, auch wenn ich mir gewünscht hätte, das Programm anschaulicher und schöner zu gestalten. Mir wurde allerdings schnell bewusst, dass dies in der kurzen Zeit nicht möglich war und habe diesen Wunsch hintenan gestellt.   
Alles in allem bin ich der Meinung, ein Projekt geschaffen zu haben, was mir Freude bereitet hat, mir nützlich ist und ganz unvergessen auch so funktioniert, wie ich es mir wünsche. Dieses Projekt war meine Zeit wert und ich hoffe, in Zukunft wieder so ein Projekt erschaffen zu können.