Wichtige Schritte beim Programmieren (nicht nur mit Matlab):

Stand: 26.9.2014

1. Was ist die Fragestellung?

- Zunächst braucht man für jedes Programm eine **SPEZIFIKATION**, also eine Beschreibung, was das Programm leisten soll.
- In diesem ersten Schritt geht es noch nicht darum festzulegen wie das Problem gelöst wird, sondern nur was seine Aufgaben sind.
- Die Spezifikation kann gleichzeitig als Hilfetext dienen, den man für jedes Programm schreiben sollte.
- In Firmen oder auch in Uni-Kursen wird häufig eine Aufgabe oder Fragestellung durch den Auftraggeber, die Firmenchefin oder die am Kurs beteiligten Lehrenden vorgegeben. Die Aufgabenstellung entspricht der Spezifikation meistens nur in groben Zügen, normalerweise ist noch einiges an Festlegungen durch die Programmiererin oder den Programmierer selbst notwendig - eventuell in Absprache mit dem Auftraggeber.

2. Welche Daten werden gebraucht und welche sollen generiert werden?

- Zur Spezifikation gehört es, den **DATENFLUSS** festzulegen:
- Welche Daten werde gebraucht und wo kommen sie her? Sollen sie aus einer Datei eingelesen werden? Oder werden sie von einem anderen Programm zur Verfügung gestellt?
- Welche Daten sollen generiert werden und was soll mit ihnen geschehen? Sollen sie z.B. in eine Datei geschrieben werden? Oder werden sie von anderen Programmen weiterverwendet? Oder werden sie vielleicht nur graphisch dargestellt und sind dann überflüssig und können wieder gelöscht werden?

3. Wie untergliedert sich das Problem?

- Wenn die Fragestellung und der Datenfluss klar sind, beginnt man damit, die Aufgabe in kleinere Teilaufgaben zu untergliedern. Häufig lässt sich das Gesamtproblem in kleine Häppchen aufteilen, die jedes für sich leicht handhabbar sind und dann nur noch in die richtige Reihenfolge gebracht werden müssen. (Dieses wichtige Konzept der Programmierung bezeichnet man auch als "**Teile und herrsche**".)
- Viele dieser kleinen Häppchen kann man in verschiedenen Kontexten gebrauchen. Dann lohnt es sich, dafür ein eigenes kleines Script oder eine Funktion zu schreiben, die man immer

wieder verwenden kann.

4. Wie geht man vor, um das Problem zu lösen?

- Ziel ist es, einen **ALGORITHMUS** aufzustellen, also eine für das Problem allgemeingültige Abfolge von Vorgängen, die nacheinander ausgeführt werden sollen
- häufig hilft es, dafür ein Beispiel von Hand durchzuspielen
- Es ist eine gute Idee, sich den Algorithmus erstmal unabhängig von der Implementierung zu überlegen, am besten auf einem Stück Papier, wenn man mag graphisch als Flussdiagramm.

5. In welchem Kontext steht das Programm ?

- Bevor man richtig loslegen kann, muss man überlegen, ob man ein **SCRIPT** oder eine **FUNKTION** schreiben möchte: Sollen alle Variablen im Workspace sichtbar sein oder will man sie kapseln?
- Funktionen sind meistens die besser Wahl, wenn man sie in einem größeren Kontext verwenden möchte. Sie besitzen definierte Eingabe- und Ausgabewerte und können entsprechend ihre Hilfsvariablen nicht mit anderen Teilen des Gesamtprogramms in Konflikt geraten.
- Wie im letzten Punkt erwähnt ist es oft sinnvoll mehr als nur ein Script oder eine Funktion zu schreiben. Ein Programm dient dann dazu, die Arbeit der Hilfsfunktionen oder Scripte zu koordinieren.

6. Wie setze ich den Algorithmus in ein Programm um?

- Erst wenn alle diese Schritte abgeschlossen sind, beginnt die **IMPLEMENTIERUNG**, bei der konkret Befehle geschrieben werden.

7. Ist das Programm sicher gegenüber Fehlbedienung?

- Wenn ein Programm von anderen Personen als der Programmiererin oder dem Programmierer selber benutzt werden soll, müssen (absehbare) eventuelle Fehlbedienungen vom Programm abgefangen werden. (Und selbst wenn man das Programm nur selber verwendet. Nach ein oder zwei Jahren weiß niemand mehr so ganz genau, welche Werte in welcher Reihenfolge beim Funktionsaufruf erwartet werden.) Lieber eine Fallunterscheidung zu viel als verlorene Daten in einem aufwändigen Experiment...

8. Tut das Programm was es soll?

    - Natürlich ist der erste Schritt, dass beim Programmablauf keine roten Zeilen mehr auftauchen - damit ist man aber noch

nicht fertig.

- Bevor man ein Programm ernsthaft einsetzt, muss man es auf seine Funktion testen. Zunächst benutzt man es für einfache Beispiele, bei denen man die richtige Lösung kennt. Dann benutzt man es in seiner Standardanwendung und schaut, ob die Ergebnisse plausibel sind. Darüber hinaus sollte man sich überlegen, welche Randbereiche von Daten eventuell ein Problem darstellen könnten oder welche Arten von "Fehlbedienung" auftreten könnten.
- Bei wichtigen Programmen ist es eine gute Idee, das Programm von jemand anderem testen zu lassen, nachdem man es selber getestet hat. Oft kommen andere Leute auf andere Ideen, was schief gehen könnte.
- Falls beim Testen in irgendwelchen Fällen Schwierigkeiten auftauchen, muss zurückgegangen werden:
    - zu Schritt 6 in sehr leichten Fällen, wenn es sich lediglich um Syntax-Probleme handelt
    - zu Schritt 4, wenn z.B. bestimmte Fälle noch nicht im Algorithmus bedacht sind
    - zu Schritt 1, wenn der generelle Ansatz so noch nicht stimmt und z.B. eine andere Art von Modellierung oder Datenanalyse verwendet werden muss.
- Achtung! Beim Ändern des Programms daran denken, auch Kommentare und Programmköpfe mit zu ändern, um sie aktuell zu halten!

9. Ist das Programm gut dokumentiert?

- Schon während man das Programm schreibt, sollte man es kommentieren, damit man selber durch seinen Code steigt, wenn man später weiterprogrammieren möchte.
- Wenn das Programm fertig und getestet ist, sollte man noch einmal komplett durch die Kommentare gehen, sie vervollständigen und so schreiben, dass auch andere Leute durch sie das Programm verstehen können. (Leider wird dieser Schritt von vielen Leuten nicht gemacht, weshalb ziemlich viel Code unnötigerweise mehrfach geschrieben wird - manchmal sogar von der selben Person, die sich nicht erinnern kann, was das eigentlich alles sollte...)