Matlab Crash Course

Resources

- Matlab
- Psychtoolbox

Die GUI (Graphical User Interface)

- Command Window: ad-hoc einzelne Befehle ausprobieren
- Editor: zusammenhängende Programme schreiben
- Rote Punkte sind Breakpoints

Variablen

... werden so definiert.

```
a=1;
b='hallo';
```

- das Semikolon unterdrückt den Output. Zeilen sollten immer mit Semikolon enden, so wird das debuggen einfacher
- Gebt euren Variablen aussagekräftige **Namen** und versucht einem konsistenten Stil zu folgen. Zum Beispiel alle normalen Variablen in camel case und Vorne klein und alle Funktionen Vorne groß geschrieben
  - `camelCaseVar`
  - `underscore_var`
  - `Function`
- Variablen haben **Typen**: Typischerweise `str` = string, text, `int`= ganze Zahl, `float`= komma Zahl. Mehr Matlab Typen in der Dokumentation . Meistens ist das egal (Matlab konvertiert wo benötigt selber) aber nicht immer. Dann muss man manuell konvertieren.

```
vpNr=2
strcat('vp', vpNr) %Falsch
strcat('vp', num2str(vpNr)) %richtig
```

- Matlab hat erst kürzlich richtige strings eingeführt. Die werden mit doppelten Anführungszeichen markiert `"text"`. Einzelne Anführungszeichen machen `'chars'`, die sich zB. in Listen anders verhalten. Einige ältere Funktionen funktionieren nur mit den alten chars und nicht mit strings.
- mit `whos x` kann man den typ herausfinden
- Es gibt auch **Listen**. Die müssen aus gleichartigen Typen bestehen. In Matlab ist eine Liste synonym mit Array, Vector und 1 Dimensionaler Matrize

```
l=[99,45,67,23]
```

```
l=[99,45,67,23.345]
l=[99,45,67,"hallo"] %vorsicht, hier sind jetzt auch die Zahlen in strings
convertiert
```

- Einzelne Listen Elemente kann man so indizieren: `l(2)` gibt das zweite Element
- **Strukte** sind gut um strukturiert auf Daten zuzugreifen

```
veg(1).name='Brokoli';
veg(1).color='green';
veg(1).tastiness=5;

veg(2).name='Tomato';
veg(2).color='red';
veg(2).tastiness=9;
```

- und weitere: `cells`(ähnlich wie strukte), `objects` (strukte mit Funktionen dazu)...
- Wo immer möglich und besonders bei längeren Programmen sollte mit variablen gearbeitet werden. Das erleichtert das schnelle ändern des Programms und die Übersichtlichkeit.

```
% nicht gut
applesPerPerson=20/5;
```

- In diesem Beispiel sind die Werte 'Hardcoded'. In einem 1000 Zeilen Programm kann es sehr nervig sein diese Zeile zu finden, um die Anzahl der Äpfel oder Personen zu ändern

```
% besser
apples=20;
people=5;
applesPerPerson=apples/people;
```

- Noch ein Typ: **Booleans** oder in Matlab auch **logicals**
  - immer True oder False. Äquivalent mit 1 und 0.
  - Eine Zahl in Bool umgewandelt ist immer True außer der 0.
  - Zusammenfügen: hierbei helfen häufig klammern
    - und `&` = ist beides true?
    - oder `|` = ist eins von beiden true?
    - z.B. `true & false` -> false, `true & true` -> true,
  - Die folgenden Vergleiche ergeben immer bools
    - ist gleich `==`
    - größer/kleiner `<`, `>`, `<=`, `>=`
    - nicht `~`
    - z.B. `1==1` -> True, `3<1` -> false
    - auch `[1,2,3]==1` -> true, false, false

Conditionals und Loops

- Ausführung von Code eine Zeile nach der anderen.

- Einen Teil des Codes nur in einer Bedingung ausführen (**IF**):

```
a=1
if a==1
    a=a+3;
else
    a=9;
end
disp(a)
```

- if statements beruhen auf Bools. Die Condition wertet immer ein Bool aus und führt den Code aus wenn es `True` ergibt. In diesem Fall ist a=1 und 1=true also ist a=True also geht auch `if a`.
- Guter code hat keine wiederholten Zeilen. Stattdessen loops (**for** und **while**) :

```
% nicht
disp(5*2)
disp(4*2)
disp(2*2)
disp(7*2)

% sondern
for i=[5,4,2,7]
    disp(i*2)
end

%oder
l=[5,4,2,7];
i=1;
while i<=length(l)
    disp(l[i])
    i=i+1
end
```

- Meistens benutzt man in Matlab den for loop, der while loop ist nur in ganz bestimmten Situationen nützlich, z.B. wenn ich nicht weiß wie lange es dauert bis die Bedingung erreicht ist. z.B. ich checke in jedem Durchlauf ob die Augen auf dem FixKreuz sind. Sobald diese Bedingung erfüllt ist, geht das Programm weiter.
- Protip: das indentieren hilft für die Übersichtlichkeit. Mit `strg+i` bzw. `command+i` macht Matlab das sogar automatisch.