1) Einführung in das Programm *Mathematica* (leicht bearbeitete Vorlage von T. Unseld)

Um *Mathematica* kennen zu lernen, beginnt man am Anfang am besten damit, es wie einen besseren Taschenrechner zu benutzen:

Mathematica als Taschenrechner

Dein leeres erstes Dokument besteht zuerst einmal aus leeren **Inputzellen**, in denen man Rechnungen eintippen kann:

```
3 + 7
```

Die oben eingegebene Rechnung ist für *Mathematica* nur Text. Sie muss mit Hilfe der **Eingabetaste** ⌅
(oder **Shift Return**!) an das eigentliche Rechenprogramm, den sogenannten **Kernel**, abgeschickt werden. Er meldet dann das Resultat in einer **Outputzelle**:

```
10
```

Man kann mehrere Rechnungen in einer Inputzelle verarbeiten lassen. Man geht für die 2. Rechnung mit Hilfe der
Returntaste auf eine neue Zeile:

```
3 * 5
5 ^ 4
```

```
15

625
```

Die Resultate der zwei obigen Rechnungen in einer Inputzelle werden getrennt in zwei Outputzellen angezeigt.
Man kann **Rechnungen in derselben Zeile** auch durch **Strichpunkte** trennen:

```
5.4 - 2.3; 4 ^ 2
3 * 5;
4 - 5
```

```
16

-1
```

Alle 4 Rechnungen wurden vom Kernel ausgeführt, jedoch nur 2 Resultate in der Outputzelle angezeigt.
Natürlich ist das obige Vorgehen nicht sehr sinnvoll, da man ja das berechnete Resultat sehen will. Später machen "unterdrückte Resultate" jedoch Sinn!

> **Der Strichpunkt in der Inputzelle unterdrückt die Ausgabe der Rechnung in der Outputzelle!**

Natürlich kann *Mathematica* auch mit Brüchen rechnen:

```
(2 / 3) ^3
(2 / 5) ^200
```

$$\frac{8}{27}$$

$$1\,606\,938\,044\,258\,990\,275\,541\,962\,092\,341\,162\,602\,522\,202\,993\,782\,792\,835\,301\,376 / 62\,230\,152\,778\,611\,417\,071\,440\,640\,537\,801\,242\,405\,902\,521\,687\,211\,671\,331\,011\,166\,147\,896 \div 988\,340\,353\,834\,411\,839\,448\,231\,257\,136\,169\,569\,665\,895\,551\,224\,821\,247\,160\,434\,722\,900\,390 \div 625$$

Wie man oben sieht, rechnet *Mathematica* solange algebraisch exakt wie es geht: das zweite Resultat ist ein Bruch!
Und das ist bei den obigen natürlichen Zahlen kein Problem.

> Wenn man will, dass *Mathematica* **numerische, gerundete Werte** ausgibt, gibt man eine der Zahlen in der
> Berechnung als Dezimalzahl ein: **2.0 anstelle von 2!**

Mathematica rechnet dann mit einer einstellbaren Genauigkeit, die am Schluss der Kopien erklärt wird:

```
(2.0 / 3) ^3
(2 / 5.0) ^200
```

0.296296

$$2.58225 \times 10^{-80}$$

Wenn man will, dass *Mathematica* exakt rechnet und erst am Schluss ein gerundetes Ergebnis liefert, benutzt man **einen ersten *Mathematica*-Befehl**:
N[z] liefert die **Zahl z auf 6 wesentliche Ziffern gerundet**: (Die **eckigen Klammern** sind **"alt" + "5"** und **"alt" + "6"**!)

```
N[(2 / 3) ^3]
N[(2 / 5) ^200]
```

0.296296

$$2.58225 \times 10^{-80}$$

Falls man das Ergebnis genauer gerundet haben will, kann man das als **2. Argument** in der Funktion **N[Zahl, wesentliche Ziffern]** eingeben:

```
N[(2 / 3) ^3, 3]
N[(2 / 3) ^3, 13]
N[(2 / 3) ^3, 50]
```

0.296

0.2962962962963

0.29629629629629629629629629629629629629629629630

Quadratwurzeln gibt man mit dem 2. *Mathematica*-Befehl **Sqrt[ ]** ("Squareroot") ein:

```
Sqrt[2]
Sqrt[2.0]
N[Sqrt[2], 20]
```

$\sqrt{2}$

```
1.41421
```

```
1.4142135623730950488
```

Wie man oben sieht, rechnet es algebraisch exakt. Das heisst, es lässt die Wurzeln stehen, ausser man benutzt Dezimalzahlen oder "numerische" Zahlen.
Dazu zwei weitere Beispiele:

```
Sqrt[2]^3
Sqrt[2]/5
```

$2\sqrt{2}$

$$\frac{\sqrt{2}}{5}$$

Wenn man für Wurzeln nicht den Sqrt[ ]-Befehl benutzen will, sondern sie wie aus der Mathematik bekannt mit $\sqrt{\ }$ in der Input-Zelle eintippen will, benutzt man eine der Paletten im Menü "Palettes" oder den Kurz-Befehl "ctrl" + "2" (gleichzeitig!).
Die Wurzeln in der folgenden Input-Zelle habe ich so eingetippt:

$\sqrt{12}$
$\sqrt{12.0}$

$2\sqrt{3}$

```
3.4641
```

Natürlich kann *Mathematica* auch den Sinus eines Winkels berechnen:

```
Sin[23]
Sin[23.0]
```

```
Sin[23]
```

```
-0.84622
```

Wie bei Wurzeln lässt *Mathematica* mathematisch exakt die Sinuswerte stehen, wenn man keine Dezimalzahlen benutzt.
Wer jedoch den obigen Sinus des Winkels 23° genau betrachtet, merkt, dass was falsch sein muss, da der Sinus von 23° sicher nicht negativ sein kann!

> Das **Argument** "Winkel" bei den **Trigonometrie-Funktionen** Sin[Winkel], Cos[Winkel] und Tan[Winkel] versteht *Mathematica* immer als **Winkel im Bogenmass**!

Oben hat er also den Sinus des Winkels 23 im Bogenmass berechnet, und der kann auch negativ sein!

Aber natürlich kann man ***Mathematica* dazu zwingen den Winkel 23 als Winkel im Gradmass zu interpretieren**:
Man tippt die Sorte "°" zusätzlich ins Argument der Funktion ein:
(Tippe "°" mit Hilfe einer der Paletten ein, oder benutze Shift bei einer der Tasten oben links auf deiner Tastatur!)

```
Sin[23 °]
Sin[23.0 °]
```

```
Sin[23 °]
```

```
0.390731
```

Die "Umkehrfunktion" des Sinus, die aus einem gegebenen Sinus-Wert einen der Winkel mit diesem Sinus berechnet, ist auf deinem Taschenrechner die $\sin^{-1}$ –Taste. In der Mathematik heisst sie jedoch "**Arc**us **Sin**us". Darum ist der *Mathematica*-Befehl dafür:

```
winkel = ArcSin[0.2]
```

```
0.201358
```

Ich habe oben den von *Mathematica* berechneteten **Winkel 0.201358 im Bogenmass** in der Variablen "winkel" mit Hilfe der Zuordnung "=" abgespeichert. Damit kann ich nun schnell nachprüfen, ob der berechnete Winkel wirklich einen Sinus-Wert von 0.2 hat:

```
Sin[winkel]
```

```
0.2
```

> Auch die Arcus-Funktionen ArcSin[ ], ArcCos[ ] und ArcTan[ ] liefern als Resultat einen Winkel im Bogenmass!

Meistens gibt man ja die Winkel im Bogenmass als Vielfache von π an. Natürlich kennt *Mathematica* die Konstante π auch:
Entweder benutzt man dafür eine der Paletten, den Kurzbefehl "alt" + "P" oder die ***Mathematica*-Konstante Pi**, mit der ich die unteren Berechnungen eingetippt habe:

```
Pi
Pi / 3
Pi / 3.0
N[Pi, 40]
```

```
π
```

$$\frac{\pi}{3}$$

```
1.0472
```

```
3.1415926535897932384626433832795028841971
```

Wie man oben sieht, lässt *Mathematica* wiederum π stehen, wenn man nicht Dezimalzahlen oder den N[ ]-Befehl benutzt.
Wenn man nun den Sinus des Winkels 30° berechnen will, hat man die folgenden 2 Varianten:

```
Sin[30 °]
Sin[Pi / 6]
```

$\frac{1}{2}$

$\frac{1}{2}$

Wie schon oben gesagt, liefert der Arcus Sinus leider den Winkel im Bogenmass, wenn der Sinus $\frac{1}{2}$ gegeben ist:

```
ArcSin[1 / 2]
```

$\frac{\pi}{6}$

Häufige Fehler beim Programmieren mit *Mathematica* verhindert folgender Merksatz:

> *Mathematica*-Befehle wie **S**in**[ ]** beginnen immer mit einem **Grossbuchstaben**, und die Argumente der Befehle befinden sich innerhalb von **eckigen Klammern**!

Noch schnell eine Bemerkung zum **Befehl N[ ]**:
Häufig wird dieser Befehl erst **nach dem Wert mit Hilfe von "//"** eingetippt:

```
2 / 3 // N
Pi // N
```

```
0.666667

3.14159
```

Aufgabe 1:
Berechne exakt und auf 6 wesentliche Ziffern gerundet:
a) $7+3\pi$ b) $2^{\pi+1}$ c) cos (5) und cos (5°) d) $-\cos^8(0)$ e) $\cos^2(1.25) + \sin^2(1.25)$
f) $\tan^3(\frac{\pi}{3})$ g) $\tan^2(30°)$ h) $\tan^{-1}(1)$ i) $\sqrt{2+\sqrt{2}}$ k) $\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}$ l) $\left(\sqrt{2}\right)^{400}$

Aufgabe 2:
a) Wie viele Ziffern hat die Zahl 5^{1234}? (Tipp: Wissenschaftliche Schreibweise!)
b) Wieviele Nullen nach dem Komma hat die Zahl $\left(\frac{1}{7}\right)^{1234}$?
c) Bestimme die letzten 2 Ziffern von 7^{1200}.
d) Bestimme die 1100. Ziffer nach dem Komma von π. (Achtung: N[ ] rundet die letzte Ziffer!)

Aufgabe 3:
Wurzeln und Sinus-Werte sind meistens irrationale Zahlen. Natürlich ist auch π irrational. Nun könnte man meinen, dass die unteren Kombinationen dieser Zahlen überraschenderweise eine natürliche Zahl ergeben. Berechne die unteren Zahlen, und zeige dann, dass sie sich doch von der erhaltenen natürlichen Zahl unterscheiden:
a) sin ($2017\sqrt[5]{2}$) ($\sqrt[5]{2}$ ist die fünfte Wurzel aus 2, also die Zahl, die hoch 5 gerechnet, 2 ergibt.)
b) $\sqrt[3]{e^{\pi\sqrt{163}}-744}$ **(**Die Zahl e ist die wichtigste Konstante in der Mathe und wird via Palette oder mit E eigetippt.)

Algebra mit *Mathematica*

Natürlich kann man in *Mathematica* auch **Werte in Variablen abspeichern**. Das geschieht mit dem **Zuordnungszeichen "="**.
Unten wird der Variable a der Wert 13 zugeordnet und dann damit Berechnungen durchgeführt:

```
a = 13
a^2
Sqrt[a 2]
```

```
13
```

```
169
```

$\sqrt{26}$

> Oben in der Wurzel wurde das Multiplikationszeichen "*" zum ersten Mal nicht mehr gebraucht. Es wurde aber ein **Leerzeichen** dafür gesetzt!

Ohne Leerzeichen versteht *Mathematica* die Variable "a2" darunter, und diese hat noch keine Belegung:

```
Sqrt[a2]
```

$\sqrt{a2}$

Nun weitere Berechnungen mit der Variable a als Elemente einer **Liste**, die in *Mathematica* mit **geschweiften**
Klammern geschrieben werden:
(Die **geschweiften Klammern** sind auf der Tastatur **"alt" + "8"** und **"alt" + "9"**!)

```
{a + 2, a^2 - 3 a, Pi^a}
```

$\{15, 130, \pi^{13}\}$

Natürlich kann man auch ganze Wörter als Variablennamen nehmen:

```
anzahl = 23; preis = 1.2; fix = 20;
anzahl preis - fix
```

```
7.6
```

Die Definitionen der Variablen wurden oben mit ";" unterdrückt.

> **Schreibe deine eigenen Variablen immer klein**, damit sie sich von den *Mathematica*-Befehlen, die immer mit einem Grossbuchstaben beginnen, klar unterscheiden!

Mathematica kann auch symbolisch rechnen:
Was macht es bei der ersten Binomischen Formel?

```
(a + b) ^2
```

$(13 + b)^2$

Die Variable a ist immer noch mit dem Wert 13 vom Anfang des Kapitels belegt. Das wäre sogar noch der Fall, wenn man das Notebook schliessen und wieder öffnen würde (Versuche es!).
Also muss man zuerst die Belegung von a löschen, wenn man symbolisch rechnen will. Das

geschieht mit "**=.**", der
Kurzversion von **Remove[a]**:

```
a =.
(a + b) ^ 2
```

$(a + b)^2$

> Wenn man nicht mehr so genau weiss, welche Variablen man schon definiert hat, benutzt man den **Befehl** `Remove["Global`*"]` , der **alle selber definierten Variablen löscht!**

```
a = 10
b = 12
Remove["Global`*"]
(a + b) ^ 2
```

```
10

12

(a + b)^2
```

Beim symbolischen Rechnen ist ein Produkt für *Mathematica* vielfach einfacher als die ausgerechnete Version. Wenn man die Binomische Formel ausgerechnet haben will, benutzt man den Befehl **Expand[ ]**:

```
Expand[(a + b) ^ 2]
```

$a^2 + 2\,a\,b + b^2$

Wenn man Ausdrücke vereinfacht haben will, benutzt man den Befehl **Simplify[ ]**:

```
Simplify[a ^ 2 - 2 a b + b ^ 2]
```

$(a - b)^2$

Die obigen 2 Befehle werden wie bei N[ ] meistens mit Hilfe von "//" erst nach dem Argument eingetippt:

```
x =.

(x + 2) ^ 3 // Expand
3 x + x ^ 2 // Simplify
```

$8 + 12\,x + 6\,x^2 + x^3$

$x\,(3 + x)$

Jetzt noch eine kleine Nebenbemerkung, wenn man nicht mit Variablen arbeiten will:
Zuerst eine normale Berechnung:

```
12 Sin[2.0]
```

```
10.9116
```

Nun möchte ich mit dem Wert 10.9116 eine weitere Berechnung machen: Ich will ihn verdoppeln. Natürlich wäre es möglich gewesen, dass ich das obige Resultat in einer Variablen abgespeichert und damit dann weiter gerechnet hätte. Hier will ich jedoch die **Variable "%"** benutzen, in der ***Mathematica*** **automatisch immer die letzte Outputzelle abspeichert:**

```
2 %
```

```
21.8231
```

Jetzt das obige Resultat mit 3 potenziert:

```
%^3
```

```
10 393.3
```

Aufgabe 5:
Berechne / "Vereinfache" die Terme: (Prüfe die Ergebnisse mit deinen Algebra-Kenntnissen nach!)
a) $(c^2+1)^4$ b) $(c^2\,de^3)^4$ c) $(\frac{1}{a}-a)^2+(\frac{1}{a}+a)^2$

Aufgabe 6:
Zerlege die Terme in Faktoren: (Prüfe die Ergebnisse mit deinen Algebra-Kenntnissen nach!)
a) $x^4-2x^3+x^5$ b) x^3+3x^2+3x+1

■ Das Lösen von Gleichungen

Mathematica löst **Gleichungen** mit dem Befehl **Solve[ ]**:

```
Solve[2 x + 10 == 20]
```

```
{{x → 5}}
```

> Das **Gleichheitszeichen** ist **"= ="** , damit es sich von der **Zuordnung** **"="** unterscheidet!

Weil oben für *Mathematica* klar war, nach welcher Variablen die Gleichung aufgelöst werden muss, brauchte der Solve-Befehl kein zweites Argument. Aber eigentlich benutzt man beim Solve-Befehl immer ein zweites Argument, mit dem man *Mathematica* mitteilt, nach welcher Variablen man aufgelöst haben will:

```
Solve[2 x + 10 == 20, x]
```

```
{{x → 5}}
```

Das Obige macht jedoch erst dann Sinn, wenn Parameter-Gleichungen gegeben sind, die *Mathematica* auch lösen kann:

```
Solve[a x + b == c, b]
Solve[a x + b == c, x]
```

```
{{b → c - a x}}
```

$$\left\{\left\{x \to \frac{-b+c}{a}\right\}\right\}$$

Wen man Parameter-Gleichungen lösen will, muss man aufpassen, dass die Parameter gelöscht und nicht mit Werten belegt sind. Wir hatten oben also "Glück", dass weder die Variable x, noch die Variable a oder b vorher mit Werten belegt wurden!
Darum benutzt man **beim Solve-Befehl vorsichtigerweise immer zuerst ein Remove["Global`*"], damit die Belegungen der Variablen gelöscht werden!** Man geht also wie folgt vor, wenn man Gleichungen löst:

```
Remove["Global`*"]
Solve[a x + b x + c == 0, x]
```

$$\left\{\left\{x \to -\frac{c}{a+b}\right\}\right\}$$

Solve löst die Gleichungen exakt, wenn es möglich ist:

```
Solve[x^6 - 2 x^4 + 7 x^2 - 12 == 0, x]
```

$$\left\{\left\{x \to -\sqrt{\frac{2}{3}-\frac{17}{3\left(107+9\sqrt{202}\right)^{1/3}}+\frac{1}{3}\left(107+9\sqrt{202}\right)^{1/3}}\right\},\ \left\{x \to \sqrt{\frac{2}{3}-\frac{17}{3\left(107+9\sqrt{202}\right)^{1/3}}+\frac{1}{3}\left(107+9\sqrt{202}\right)^{1/3}}\right\},\right.$$

$$\left\{x \to -\sqrt{\left(\frac{2}{3}+\frac{17}{6\left(107+9\sqrt{202}\right)^{1/3}}-\frac{17\,i}{2\sqrt{3}\left(107+9\sqrt{202}\right)^{1/3}}-\frac{1}{6}\left(107+9\sqrt{202}\right)^{1/3}-\frac{i\left(107+9\sqrt{202}\right)^{1/3}}{2\sqrt{3}}\right)}\right\},$$

$$\left\{x \to \sqrt{\left(\frac{2}{3}+\frac{17}{6\left(107+9\sqrt{202}\right)^{1/3}}-\frac{17\,i}{2\sqrt{3}\left(107+9\sqrt{202}\right)^{1/3}}-\frac{1}{6}\left(107+9\sqrt{202}\right)^{1/3}-\frac{i\left(107+9\sqrt{202}\right)^{1/3}}{2\sqrt{3}}\right)}\right\},$$

$$\left\{x \to -\sqrt{\left(\frac{2}{3}+\frac{17}{6\left(107+9\sqrt{202}\right)^{1/3}}+\frac{17\,i}{2\sqrt{3}\left(107+9\sqrt{202}\right)^{1/3}}-\frac{1}{6}\left(107+9\sqrt{202}\right)^{1/3}+\frac{i\left(107+9\sqrt{202}\right)^{1/3}}{2\sqrt{3}}\right)}\right\},$$

$$\left.\left\{x \to \sqrt{\left(\frac{2}{3}+\frac{17}{6\left(107+9\sqrt{202}\right)^{1/3}}+\frac{17\,i}{2\sqrt{3}\left(107+9\sqrt{202}\right)^{1/3}}-\frac{1}{6}\left(107+9\sqrt{202}\right)^{1/3}+\frac{i\left(107+9\sqrt{202}\right)^{1/3}}{2\sqrt{3}}\right)}\right\}\right\}$$

Wenn man die numerisch gerundeten Lösungen der Gleichung haben will, kann man nicht einfach den N[ ]-Befehl nehmen, da die Lösungen in einer "komischen" Liste "versteckt" sind.
Für dieses Problem gibts den **NSolve-**Befehl, der direkt die **numerisch gerundeten** Lösungen meldet:

```
NSolve[x^6 - 2 x^4 + 7 x^2 - 12 == 0, x]
```

```
{{x → -1.34352}, {x → -1.15669 - 1.11376 i}, {x → -1.15669 + 1.11376 i},
  {x → 1.15669- 1.11376 i}, {x → 1.15669+ 1.11376 i}, {x → 1.34352}}
```

Mathematica meldet 6 verschiedene Lösungen, 4 davon jedoch mit einem "komischen" i. Diese 4 Lösungen gibts gar nicht, da i keine reelle Zahl ist: es ist die Zahl, die quadriert -1 ergibt, also eigentlich $\sqrt{-1}$:

```
√-1
```

```
i
```

Diese nicht-existierenden negativen Wurzel haben den Mathematikern geholfen, Formeln für Gleichungen zu finden und
haben heute in vielen naturwissenschaftlichen Bereichen eine grosse Bedeutung. Darum sind sie auch Algebra-Stoff in der
4. Klasse.
Die obige Gleichung hat also nur die 2 (reellen) Lösungen x = ± 1.34352.

> Nur die **Lösungen ohne i** beim Solve-Befehl sind uns momentan interessierende reelle Lösungen der Gleichungen!

Der Solve- und der NSolve-Befehl kann leider nicht alle Arten von Gleichungen lösen:

```
Solve[Tan[x] == x, x]
NSolve[Tan[x] == x, x]
```

```
Solve[Tan[x] == x, x]
```

```
NSolve[Tan[x] == x, x]
```

Solve[ ] und NSolve[ ] können nur sogenannte "algebraische" Gleichungen lösen, die obige "transzendente"
Gleichung hingegen nicht, weil es dafür keine Lösungs-Formeln und -Verfahren gibt.
Es braucht numerische Verfahren, die eine Lösung annähern können! Das macht *Mathematica*

beim FindRoot-Befehl:

```
FindRoot[Tan[x] == x, {x, 1}]
```

$\{x \to 2.63008 \times 10^{-8}\}$

Die offensichtliche Lösung x = 0 findet FindRoot[ ] wenigstens annähernd.
{x, 1} bedeutet, dass das Verfahren bei x = 1 auf die Suche nach einer Lösung x geht.
FindRoot[ ] findet also nur **eine** Lösung der Gleichnung. Zum Beispiel findet das Verfahren eine andere Lösung, wenn man mit dem Startwert x = 10 beginnt:

```
FindRoot[Tan[x] == x, {x, 10}]
```

$\{x \to 10.9041\}$

Aufgabe 7:
Bestimme die exakten Lösungen der Gleichungen:

a) $2x^2 + 26x = 136$

b) $ax + b = x$ (a, b Parameter)

c) $\sqrt{x^2 + 9} = 3\sqrt{x - 1}$

d) $(s - 1)\left(st^2 + 1\right) = st\,(s - t)$ zuerst nach s und dann nach t auflösen.

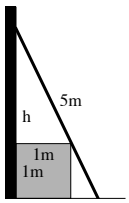
Aufgabe 8:
Bestimme die numerisch gerundeten Lösungen der Gleichungen:

a) $36x^4 - 25x^2 + 4 = 0$

b) $x^3 + 2x^2 + 4x + 8 = 0$

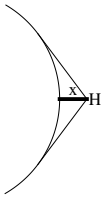
c) $\cos^2(x) = x^2$

Aufgabe 9: (freiwillige Aufgabe!)



Welche maximale Höhe h kann man mit einer 5m langen Leiter an einer senkrechten Wand erreichen, wenn der Zugang durch ein Podest von 1m Höhe und 1m Tiefe versperrt ist?

Aufgabe 10: (freiwillige Aufgabe mit einem erstaunlichen Resultat!)



Um die Erde (Kugel mit Radius r = 6370 km) wird straff ein Seil gespannt. Mike verlängert nun das Seil um 1 m und zieht es mit seiner Hand an einer Stelle weg von der Oberfläche. Wie weit weg von der Oberfläche ist seine Hand, wenn das Seil wieder straff wird?

Die meisten *Mathematica*-Befehle kann man auch als Funktionen verstehen:
Zum Beispiel ordnet der *Mathematica*-Befehl Sqrt[*x*] jedem *x* eindeutig die Wurzel von *x* zu:
$x \longrightarrow \sqrt{x}$.
Also ist y = Sqrt[*x*] eine Funktion mit dem Namen "Sqrt"; in der Mathematik nennt man sie Wurzelfunktion $y = \sqrt{x}$.

Wenn ich die Funktionswerte y der x-Werte x = 0, x = 2, x = 4, x = 5, x = 6.2 haben möchte, könnte ich sie wie folgt als Liste der Werte programmieren:

```
{Sqrt[0], Sqrt[2], Sqrt[4], Sqrt[5], Sqrt[6.2]}
```

$\{0, \sqrt{2}, 2, \sqrt{5}, 2.48998\}$

Jetzt soll gezeigt werden, wie man in *Mathematica* **eigene Funktionen, das heisst eigene Befehle definiert:**
Ich möchte zum Beispiel einen Befehl kreieren, der jeden Wert x quadriert. Ich nenne diesen Befehl "quadrieren":

```
quadrieren[x_] := x^2
```

Obwohl die obige Zelle an den Kernel geschickt wurde, erscheint keine Outputzelle, weil eine sogenannte **verzögerte Zuordnung ": ="** benutzt wurde. Das macht auch Sinn, da x nicht x^2 zugeordnet werden soll, sondern der Befehl "quadrieren" soll erst dann zum Zug kommen, wenn **später** für x_ Werte eingegeben werden:

```
quadrieren[3]
{quadrieren[-2], quadrieren[Pi], quadrieren[2 / 3]}
```

9

$\left\{4, \pi^2, \frac{4}{9}\right\}$

"x_" nennt man Muster oder **Pattern** und hat nichts mit der Variablen x zu tun:
Es wird **irgendein Ausdruck x_** , der als Argument in den Befehl eingegeben wird, quadriert:

```
{quadrieren[a], quadrieren[2 a], quadrieren[3 a^3 + 1], quadrieren["Velo"]}
```

$\left\{a^2, 4\,a^2, \left(1 + 3\,a^3\right)^2, \text{Velo}^2\right\}$

Wie man oben sieht, wurde sogar der Text "Velo" quadriert. "Velo" ist also kein Variablenname, sondern wird wegen den Anführungs- und Schlusszeichen von *Mathematica* nur als Text behandelt. Eine **Zeichenkette** wie "Hallo0815Velo" nennt man in der Informatik einen **String**.

Zur Veranschaulichung des obigen neuen Vorgehens möchte ich nun einen zweiten eigenen Befehl (eigene Funktion) definieren, der bei Eingabe des Radius des Kreises den Flächeninhalt berechnet. Mein Befehl, meine Funktion soll den Namen "flaeche" haben:

```
flaeche[r_] := r^2 * Pi
```

Mit dem obigen Befehl kann man für den Pattern r_ irgendeinen Radius eingeben, und es wird die entsprechende Fläche mit der Variablen r berechnet:

```
flaeche[3]
flaeche[2.3]
```

$9\,\pi$

16.619

Nun möchte ich meinen Befehl so verbessern, dass er immer eine numerisch gerundete Fläche berechnet, weil ich mir oben unter $9\,\pi$ als Flächeinhalt nichts vostellen kann. Dieser neue Befehl, diese neue Funktion soll nur noch den Namen "f" haben:

```
f[r_] := N[r^2 Pi, 4]
```

Die oben definierte Funktion "f" sollte die "Fläche auf 4 wesentliche Ziffern gerundet" liefern:

```
f[3]
```

```
28.27
```

Eine **eigene Funktion** (einen eigenen Befehl) mit dem **Namen "mike"**, die jedem t zum Beispiel den Wert $2 \cdot t - 3$ zuordnet, definiert man in *Mathematica* nach folgender Struktur:

```
mike[t_] := 2 * t - 3
```

Benutze bei Funktionsdefinitionen die **verzögerte Zuordnung ":="** und den **Pattern "t_"** links vom ":="!
Denke daran, dass rechts vom ":=" kein Pattern "t_" mehr steht, sondern nur noch "t"!

Benutze **für deine eigenen Funktionsnamen nur Kleinbuchstaben**, damit sie sich von den *Mathematica*-Befehlen, die immer mit einem Grossbuchstaben beginnen, unterscheiden!

Man kann auch **Befehle, Funktionen mit mehreren Argumenten** definieren:

```
produkt[a_, b_, c_] := a * b * c
produkt[2, 3, 4]
produkt[x^2, 2 x^3, 4 c]
```

```
24
```

$$8\,c\,x^5$$

Aufgabe 11:
Definiere eine Funktion, die
a) bei Eingabe des Radius r den Umfang des entsprechenden Kreises bestimmt, und teste deine Funktion mit ein paar Radien.
b) bei Eingabe der Katheten eines rechtwinkligen Dreiecks die Hypotenuse berechnet, und teste deine Funktion mit ein paar Werten.
c) bei Eingabe von 4 Noten den gerundeten Durchschnitt berechnet , und teste deine Funktion mit ein paar Noten.
(Tipp: Damit deine Funktion auch bei Dezimallzahlen als Noten klappt: Benutze nicht N [ ], sondern
Round[ ]! Der Unterschied zwischen den 2 Befehlen wird am Schluss dieser Kopien genauer erklärt!)
d) bei Eingabe der Koordinaten von 2 Punkten P und Q in der Ebene den gerundeten Abstand von P zu Q
berechnet. Teste deine Funktion mit ein paar Punkten.

Aufgabe 12:
a) Die Note einer Prüfung berechnet sich gemäss der Formel

$$\text{Note} = \frac{\text{erreichte Punktzahl}}{\text{maximale Punktzahl}} \cdot 5 + 1$$

Schreibe eine Funktion note, die für eine Prüfung, bei der es 14 Punkte gab, die Noten berechnet. Dabei soll die
berechnete Note auf zwei wesentliche Stellen gerundet werden. Teste deine Funktion mit ein paar Punkten.

b) Bei den Exen ist die Maximalpunktzahl nicht immer gleich. Schreibe eine zweite Funktion note,

bei der als zweites Argument auch die Maximalpunktzahl übergeben werden kann. Teste deine Funktion mit

ein paar konkreten Punktzahlen.

Aufgabe 13:

a) Der *Body-Mass-Index* berechnet sich aus Gewicht g (in kg) und Grösse h (in m) einer Person gemäss

der Formel: $\text{bmi} = \frac{g}{h^2}$

Schreibe eine Funktion, die aus Gewicht und Grösse den BMI berechnet.

b) Schreibe eine Funktion, die aus der Zeit t, Beschleunigung a und Anfangsgeschwindigkeit V_0 den zurückgelegten

Weg s eines gleichmässig beschleunigten Körpers berechnet, und teste deine Funktion mit ein paar Werten.

Die zugehörige Formel lautet: $s = \frac{a}{2} t^2 + V_0 t$

Rechengenauigkeit und der N[ ]-Befehl

Mathematica rechnet exakt, wenn es geht. Zum Beispiel sind Brüche für ihn Brüche und keine gerundeten
Dezimalzahlen:

```
exakt = 1 / 3
```

$$\frac{1}{3}$$

Also kann er die Variable "exakt" mit jeder beliebigen Genauigkeit gerundet darstellen :

```
N[exakt, 20]
N[exakt, 50]
```

```
0.33333333333333333333
```

```
0.33333333333333333333333333333333333333333333333333
```

Der Befehl `N[z, k]` ist ein **Formatierungsbefehl, der versucht die Zahl z auf k wesentliche Ziffern gerundet darzustellen.** Er rundet die Zahlen im Kernel eigentlich nicht, sondern stellt die Zahl gerundet dar.
Zum Runden gibts darum den Befehl Round[ ]!

Etwas schwieriger zu beurteilen ist, wie exakt Mathematica bei Dezimalzahlen rechnet!
Ein erstes Beispiel:

```
2.123456789
```

```
2.12346
```

Die obige Dezimalzahl wird in der Outputzelle bereits auf 6 wesentliche Ziffern gerundet **dargestellt**!
Auch mit dem Befehl N[ ] kann man die Darstellung der Dezimalzahl nicht mehr verändern:

```
N[2.123456789, 8]
N[2.123456789, 2]
```

```
2.12346
```

```
2.12346
```

Die auf **6 wesentliche Stellen gerundete Darstellung** ist bereits durch das Setzen des Dezimalpunkts festgelegt.
Natürlich ist die obige Zahl intern immer noch exakt gespeichert. Man kann das mit dem Befehl **Precision[ ]**
nachprüfen, der angibt, mit wie vielen Ziffern der Wert intern gespeichert ist:

```
Precision[2.123456789]
```

```
MachinePrecision
```

```
$MachinePrecision
```

```
15.9546
```

Die Maschinengenauigkeit ist ungefähr auf 16 Ziffern eingestellt. Der obige ungenaue Wert entsteht, weil *Mathematica* intern nicht im Dezimalsystem, sondern im Binärsystem mit den Ziffern 0 und 1 mit 53 Ziffern rechnet:
(Die untere Zeile kannst du noch nicht verstehen, weil du den Zehnerlogarithmus noch nicht kennst!)

```
Log[10, 2^53] // N
```

```
15.9546
```

N[z] ist also nur ein Darstellungsbefehl, der die Zahl z in der Output-Zelle gerundet darstellt und hat darum seine "Macken".
Wenn man nicht nur die Darstellung gerundet haben will, benutzt man besser den Round-Befehl, der wirklich die Zahl rundet:

```
zahl1 = 6.4567
Round[zahl1]
zahl2 = 0.4567
Round[zahl2]
```

```
6.4567
```

```
6
```

```
0.4567
```

```
0
```

Ohne 2. Argument rundet Round[ ] auf ganze Zahlen. Mit einem 2. Argument kann man angeben, auf wieviele Stellen nach dem Komma man die Zahl gerundet haben will:

```
zahl1 = 6.4567
Round[zahl1, 1 / 100]
Round[zahl1, 0.01]
```

```
6.4567
```

$$\frac{323}{50}$$

```
6.46
```

Oben wurde in beiden Fällen auf Hunderstel gerundet. Im ersten Fall stellt *Mathematica* die Zahl sogar als gerundeten Bruch dar. Wer die negativen Exponenten verstanden hat, könnte das Obige auch so programmieren:

```
Round[zahl1, 10^(-2)]
Round[zahl1, 10^(-2.0)]
```

$$\frac{323}{50}$$

```
6.46
```

Zurück zur Rechengenauigkeit und dem N[ ]-Befehl:
Wenn man Dezimalzahlen mit mehr als 16 Stellen benutzt, erhöht *Mathematica* automatisch die Genauigkeit von 16 Stellen, damit man immer noch exakt damit rechnen kann:

```
2.123456789123456789
Precision[2.123456789123456789]
```

```
2.123456789123456789
```

```
18.327
```

Wie man oben sieht, wird die Dezimalzahl sogar überraschenderweise exakt in der Outputzelle dargestellt.
Sobald man die Maschinengenauigkeit von 16 Ziffern erhöht, wird auch die auf 6 Stellen gerundete Darstellung in der Outputzelle aufgehoben.
So kann man nun auf solche Zahlen den N[ ]-Befehl anwenden:

```
N[2.123456789123456789, 10]
N[2.123456789123456789, 3]
```

```
2.123456789
```

```
2.12
```

Was passiert, wenn man die Dezimalzahl 1.5 zur obigen Zahl addiert?

```
2.123456789123456789 + 1.5
```

```
3.62346
```

```
Precision[2.123456789123456789 + 1.5]
```

```
MachinePrecision
```

Weil die Zahl 1.5 intern wegen dem Dezimalpunkt mit 16 Stellen gespeichert ist, kann die Rechnung höchstens mit 16 Stellen Genauigkeit ein Resultat liefern. Also gibt *Mathematica* das Resultat nur noch mit Maschinengenauigkeit von 16 Ziffern an, um "auf der sicheren Seite" zu sein!
Natürlich kann man die obige Rechnung exakt durchführen. Dazu später mehr.

Wie genau werden unendliche Dezimalzahlen dargestellt? Zum Beispiel der anfängliche Bruch $\frac{1}{3}$ = `0.33333....` , wenn man ihn mit einem Dezimalpunkt schreibt?

```
numerisch1 = 1.0 / 3
Precision[numerisch1]
```

```
0.333333
```

```
MachinePrecision
```

Wegen dem Dezimalpunkt wird 1.0 und damit auch die restliche Berechnung (geteilt durch 3) mit 16 Ziffern Genauigkeit durchgeführt.
Wie oben wird also der Dezimalbruch auf 6 wesentliche Ziffern gerundet dargestellt und mit 16 Ziffern intern gespeichert. `1.0 / 3` entspricht also nicht mehr dem Bruch $\frac{1}{3}$, sondern nur noch der endlichen Dezimalzahl 0.33....333.
Wegen dem Dezimalpunkt wird die **ganze** untere Berechnung mit derselben Genauigkeit und Rundung wie bei 1.0
durchgeführt. So sieht man leider keinen Unterschied zwischen der Variablen "exakt" und "numerisch1", obwohl es einen gibt:

```
1 / 3 - 1.0 / 3
```

```
0.
```

Wenn man nun genauer, zum Beispiel mit 30 Stellen Genauigkeit arbeiten will, gibt es mehrere Möglichkeiten.
Die erste Möglichkeit benutzt den N[ ]-Befehl zusammen mit dem exakten Bruch:

```
numerisch2 = N[1 / 3, 30]
```

```
0.333333333333333333333333333333
```

Intern ist der Wert der Variablen "numerisch2" nicht mehr auf 16 Stellen genau abgespeichert, sondern auf 30 Stellen genau:

```
Precision[numerisch2]
```

```
30.
```

Nun werden auch weitere Rechnungen mit der Variablen automatisch mit 30 Ziffern ausgeführt:

```
rechnung1 = 2 numerisch2^3
Precision[rechnung1]
```

```
0.074074074074074074074074074074
```

```
29.5229
```

Natürlich muss man aufpassen, dass keine weiteren Dezimalzahlen in der Berechnung vorkommen:

```
rechnung2 = 2.0 numerisch2^3
Precision[rechnung2]
```

```
0.0740741
```

```
MachinePrecision
```

Die 2. Möglichkeit benutzt den **SetPrecision-Befehl**, der den "Accent Graph" als Kurzform hat:

```
numerisch3 = 1.0`30 / 3
```

```
0.333333333333333333333333333333
```

Der Dezimalbruch 1.0 wurde oben auf 30 Stellen genau definiert, also auch die weitere Rechnung, die ihn durch 3 teilt. Also ist 0.3333.... auf 30 Stellen genau gespeichert:

```
Precision[numerisch3]
```

```
30.
```

Auch hier werden nun weitere Rechnungen mit der Variablen automatisch mit 30 Ziffern ausgeführt:

```
rechnung3 = 2 numerisch3^3
Precision[rechnung3]
```

```
0.074074074074074074074074074074
```

```
29.5229
```

Aufgabe 25:

a) Was meldet *Mathematica* bei den Berechnungen? Überlege es dir vorher!

`1 - 10^-10` ?

`1.0 - 10^-10` ?

`(1.0 - 10^-10) - (1.0 - 10^-10)` ?

b) Führe die Rechnung `1.5 + 2.123456789123456789` exakt durch.

Man kann mit bis zu 1'000'000 Stellen rechnen. Man muss sich einfach im Klaren sein, dass die Berechnungen etwas länger dauern:

```
numerisch4 = N[Sqrt[5]];
rechnung4 = numerisch4^100 // Timing
Precision[rechnung4]
```

$$\{0.000059,\ 8.88178 \times 10^{34}\}$$

```
MachinePrecision
```

```
numerisch5 = N[Sqrt[5], 1 000 000];
Precision[numerisch5]
rechnung5 = numerisch5^100; // Timing
Precision[rechnung5]
```

$1. \times 10^{6}$

```
{1.76641, Null}
```

```
999 998.
```

Es dauert ca. 30'000 Mal länger, aber immer noch ziemlich schnell, wenn *Mathematica* mit 1 Million anstelle von 16 Ziffern rechnen muss.

Natürlich muss man die Maschinen-Genauigkeit von 16 Stellen selten verändern, sondern man arbeitet mit *Mathematica* meistens exakt und stellt dann die Resultate mit dem N[ ]-Befehl gerundet dar. Man sollte sich einfach bewusst sein, dass man nur mit 16 Stellen Genauigkeit arbeitet, sobald man einen Dezimalpunkt setzt!
Bei Schleifen ist das Setzen des Dezimalpunkt manchmal auch sehr nützlich, weil man damit verhindert, dass
Mathematica algebraisch exakt rechnen muss. Damit können Schleifen viel schneller durchlaufen werden.
Als Beispiel eine Schleife mit Bruch-Operationen, bei der verschachtelte Doppelbrüche entstehen, wenn *Mathematica* exakt rechnen muss :

```
a = 3;
Do[a = a + 1 / a, {i, 1, 22}] // Timing
N[a, 40]
```

```
{47.7494, Null}
```

```
7.342996076009128585091552673043861079282
```

Es braucht einige Zeit und sogar einige Minuten, wenn man die Schleifen-Anzahl von 22 auf zum Beispiel 25 erhöhen würde, weil *Mathematica* mit "Riesen-Doppelbrüchen" rechnen muss. Um einiges schneller wird es, wenn man mit
Dezimalzahlen arbeitet: **3.0** anstelle von **3**!
Zuerst mit 16-stelligen Dezimalzahlen:

```
a = 3.0;
Do[a = a + 1 / a, {i, 1, 22}] // Timing
N[a, 40]
Precision[a]
```

```
{0.000493, Null}
```

```
7.343
```

```
MachinePrecision
```

Mathematica meldet das auf 6 wesentliche Ziffern gerundete **exakte** Resultat 7.34300 ohne die Nullen!
Jetzt mit 100 Stellen Genauigkeit:

```
a = 3.0`100;
Do[a = a + 1 / a, {i, 1, 22}] // Timing
N[a, 40]
Precision[a]
```

```
{0.00081, Null}
```

```
7.342996076009128585091552673043861079282
```

```
100.
```

Das Resultat ist "auf 40 Stellen genau" genau so exakt wie beim algebraischen Berechnen, aber die Schleife wurden ca. 50'000 Mal schneller durchlaufen!

Auf diesen Kopien zum letzten Mal am Schluss des Kapitels ein kleiner Vorgeschmack, was *Mathematica* auch kann:

Aufgabe 26:
Tippe den unteren Befehl ein, und klicke in die Graphik. So solltest du mit Hilfe des Cursors die entstandene Welle im Raum drehen können:

```
Plot3D[Sin[x], {x, 0, 12}, {y, 0, 6}, Boxed → False, Axes → False]
```

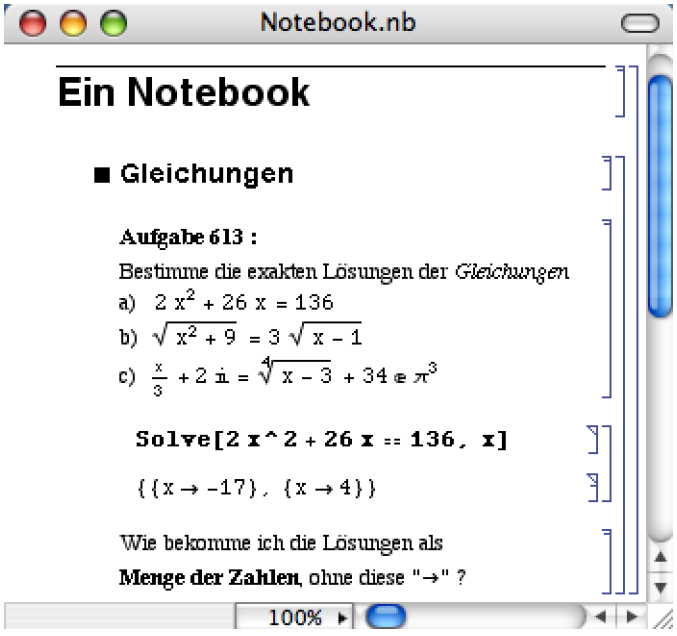
Benutzung der Notebook-Oberfläche

Mathematica-Dateien nennt man **Notebooks** ("Notizbücher"). Sie bekommen darum auch die Endung "**.nb**", wenn man sie abspeichert.
Die Notebook-Oberfläche oder **Front-End** von *Mathematica* ist mit einem Textsystem vergleichbar. Alles, was bisher in die **Inputzellen** geschrieben und in den **Outputzellen** als Resultate gemeldet wurde, war Text.
Nun gibt es **weitere Arten von Zellen**, die nicht wie die Inputzellen dazu da sind, Berechnungen an den Kernel zu schicken. Man benutzt sie, um ein Notebook zu gestalten:

Ändere den **Style** im Menü **Format** von **Input** auf zum Beispiel **Text** und die nächste Zelle, die du bearbeitest, ist keine Input-, sondern eine Textzelle. Solche Zellen können natürlich nicht mehr an den Kernel zur Verarbeitung geschickt werden. Sie sind nur dazu da, Programme mit Text zu kommentieren oder Skripte wie dieses hier zu erstellen.

Aufgabe 27:
a) Erstelle das folgende Notebook mit den verschiedenen Zellarten:
Tipp : Für das Eingeben der Gleichungen verwendet man die **Palette BasicTypesetting** im Menü **File**.

Weil man den Zelltyp **Text** sehr oft braucht, hat er einen Kurzbefehl!
Wenn man Text *kursiv* oder **fett** schreiben will, benutzt man das **Face Italic** oder **Bold** im Menü **Format**. Auch sie haben Kurzbefehle!



Zelle vom Typ :

Section

Subsection

Text

Input und Output

Text

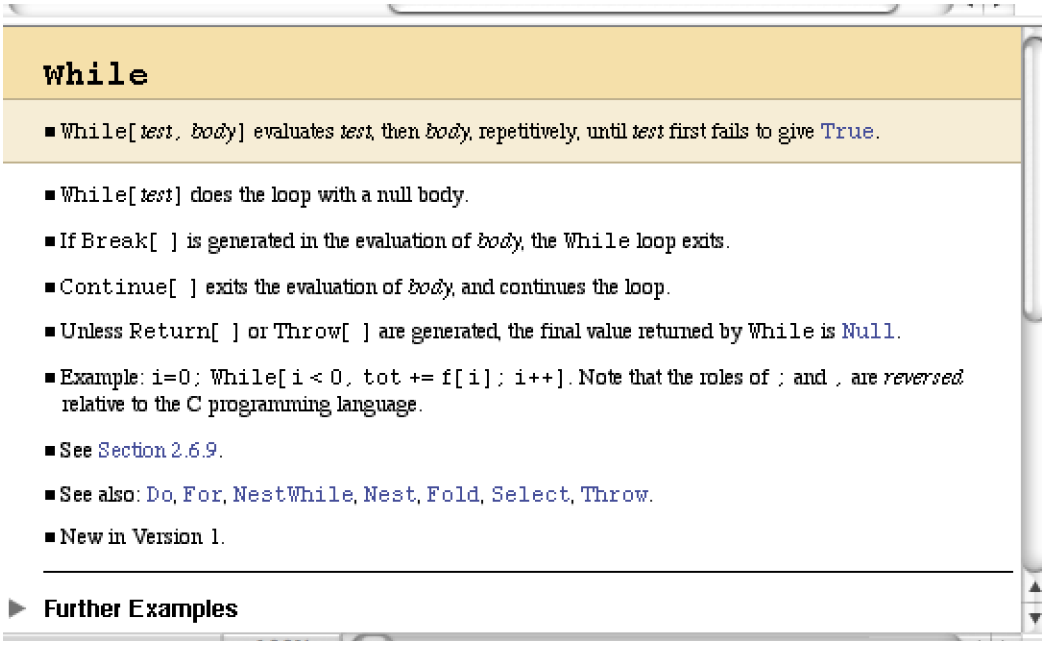
b) Füge die Kommentarzelle "Das kommt vom Kernel :" zwischen die In- und Outputzelle ein.
Tipp : Einfach den Cursor auf die horizontale Linie zwischen den Zellen setzen und klicken.
Wenn man jetzt was schreibt, steht es in einer zwischen den Zellen entstandenen neuen Zelle.

c) **Kommentare innerhalb der Inputzelle** müssen die Struktur **(*** Kommentar ***)** haben, damit sie die Eingabe an den Kernel nicht stören.
Setze so den Kommentar "stört nicht!" im Solve-Befehl zwischen das "x" und das "==", und prüfe nach, ob das die Lösung in der Outputzelle stört.

d) Drucke zum Schluss dein eigenes Notebook einmal als ganzes Notebook und einmal nur die

Aufgabe zusammen mit der Lösung aus.
Tipp : **Print Selection** im Menü **File**, wenn man nur einzelne Zellen des Notebooks drucken will.

Das Menü **Help** benötigt man oft, weil man auf die Suche nach *Mathematica*-Befehlen geht, die einem helfen ein Problem zu lösen. Oder man weiss nicht mehr so genau, wie die Struktur eines Befehls aussieht.
Zum Beispiel weiss ich nicht mehr, wie der **While-Befehl** benutzt wird. Wo kommt schon wieder die Bedingung hin?
Nun kann man zum Beispiel **Find Selected Function** im Menü **Help** wählen. Dort "While" eintippen und ein Notebook **While** erscheint. Wenn man es öffnet, erscheint folgender Text:



Meistens sind die Hilfen sehr allgemein formuliert. Am besten geht man rasch zu den Beispielen "**Further Examples"**, wenn man den Text nicht versteht! Mache es!

Dieses obige Fenster zu While erhält man schneller, wenn man in einer Inputzelle das Wort "While" schreibt, markiert und dann mit dem Kurzbefehl ⇧⌘F (alles gleichzeitig!) für "Suchen des *Mathematica*-Befehl" arbeitet.
Versuche es!

Eine andere Version wie man Hilfe findet, ist via Inputzelle und Kernel mit dem Befehl **?**:

```
? While
```

While[*test*, *body*] evaluates *test*, then *body*, repetitively, until *test* first fails to give True. ≫

Aufgabe 28:
Finde raus, wie der dir unbekannte **For**-Befehl funktioniert, und programmiere damit die 3er-Reihe bis zur Zahl 702.
Was ist eigentlich der Unterschied zum While-Befehl, oder wieso gibts eigentlich 2 Befehle?