Einführung in *Mathematica*

Tipp: Um *Mathematica* auszuprobieren gibt es eine Probeversion. Ein Teil der Funktionalität von *Mathematica* ist auch über die Webseite WolframAlpha frei verfügbar und am Raspberry Pi ist eine Gratisversion vorinstaliert.

Erste Schritte

Mathematica ist ein umfassendes Programmpaket, das sowohl symbolisch als auch numerisch rechnen kann. Im einfachsten Fall kann es wie ein Taschenrechner verwendet werden. Geben wir zum Beispiel 3 + 5 ein und drücken danach die ENTER-Taste am Ziffernblock (oder alternativ SHIFT+RETURN):

```
In[1]:= 3 + 5
Out[1]= 8
```

Ein Strichpunkt am Ende einer Anweisung unterdrückt die Ausgabe. Sie können mehrere Anweisungen auf einmal eingeben, indem Sie diese durch Strichpunkte trennen:

```
In[2]:= x = 5; 3 * x
Out[2]= 15
```

Das Multiplikationszeichen * muss nicht geschrieben werden,ein Leerzeichen genügt. Vergessen Sie aber auf dieses Leerzeichen nicht - das kann nämlich einen großen Unterschied machen, wie das folgende Beispiel zeigt:

```
In[3]:= xy + x y
Out[3]= xy + 5 y
```

`xy` ohne Leerzeichen wird also als Variable aufgefasst. Auch Groß-/Kleinschreibung wird von *Mathematica* unterschieden:

```
In[4]:= X + x
Out[4]= 5 + X
```

Sie haben bereits gesehen, dass jede Eingabe und jede Ausgabe mit einer Nummer versehen werden. Sie können auf den jeweiligen Ausdruck jederzeit zurückgreifen:

```
In[5]:= Out[1] / 2
Out[5]= 4
```

Die *unmittelbar vorhergehende* Ausgabe erhalten Sie mit einem Prozentzeichen:

```
In[6]:= % + 3
Out[6]= 7
```

Momentan ist **x** mit dem Wert 5 belegt:

```
In[7]:= 1 / (1 - x) + 1 / (1 + x)
```

$$\text{Out[7]= } -\frac{1}{12}$$

Mit **Clear** können Sie diese Belegung löschen:

```
In[8]:= Clear[x]
```

Nun ist **x** wieder unbelegt:

```
In[9]:= 1 / (1 - x) + 1 / (1 + x)
```

$$\text{Out[9]= } \frac{1}{1-x} + \frac{1}{1+x}$$

Zur Vereinfachung eines Ausdrucks können Sie **Simplify** verwenden. Vereinfachen wir beispielsweise die letzte Ausgabe:

```
In[10]:= Simplify[%]
```

$$\text{Out[10]= } -\frac{2}{-1+x^2}$$

Für hartnäckige Fälle steht auch noch die umfangreichere Variante **FullSimplify**, die sich noch mehr anstrengt, zur Verfügung.

Vielleicht ist Ihnen aufgefallen, dass *Mathematica* Ausdrücke in einer gut lesbaren Form ausgibt, also beispielsweise eine Potenz in der Form x^2 anstelle von **x^2**. Auch wir können Brüche, Potenzen usw., entweder mit den üblichen Symbolen /, ^ usw. eingeben, oder wir können die entsprechenden Symbole mit der Maus aus einer Palette auswählen. So kann etwa der Bruch

```
In[11]:= (x + 1) / x^2;
```

mit der Maus über die Palette *Basic Math Assistant* (zu finden im Menüpunkt *Palettes*) auch in der Form

```
In[12]:= (x + 1)/x^2;
```

eingegeben werden. Dafür gibt es auch eine Reihe von Tastaturkürzel: Z.B erhält man eine Bruch mit CTRL/ und die Potenz mit CTRL^. Für die meisten Symbole erfährt man das Tastenkürzel indem man mit der Maus über das entsprechende Symbol auf der Palette fährt.

Der Strichpunkt am Ende der Eingabe bewirkt, dass die Ausgabe unterdrückt wird (der Ausdruck wird aber natürlich ausgewertet und auf das Ergebnis kann mit **%** oder **Out[]** zugegriffen werden).

Hilfe zu *Mathematica*-Befehlen finden Sie im Menü unter *Help→Documentation Center* oder mit dem Befehl **?Befehl**, z.B:

```
In[13]:= ? Sin
```

Sin[*z*] gives the sine of *z*. ≫

Übung: Versuchen Sie, die Bezeichnung für die Kreiszahl π in *Mathematica* herauszufinden.

Funktionen

Mathematica kennt eine Vielzahl von mathematischen Funktionen. Diese Funktionen beginnen immer mit einem Großbuchstaben. Die Argumente werden in eckigen Klammern angegeben. Einige der eingebauten Funktionen sind:

Sqrt[*x*]	Wurzelfunktion $\sqrt{x}$
Exp[*x*]	Exponentialfunktion e^x
Log[*x*]	Natürlicher Logaritmus $\ln(x)$
Log[*a*, *x*]	Logaritmus $\log_a(x)$
Sin[*x*], Cos[*x*]	Sinus – und Kosinusfunktion
Abs[*x*]	Absolutbetrag $\lvert x \rvert$

Zum Beispiel können wir die Wurzel aus 4 berechnen:

In[14]:=
```
Sqrt[4]
```

Out[14]=
```
2
```

Wenn wir aber etwa `Sin[1]` eingeben, so erhalten wir:

In[15]:=
```
Sin[1]
```

Out[15]=
```
Sin[1]
```

Das ist vermutlich nicht das Ergebnis, das Sie erwartet haben! *Mathematica* wertet den Ausdruck hier symbolisch (und nicht numerisch) aus. Und da es für `Sin[1]` symbolisch keinen einfacheren Wert gibt, wird er unverändert ausgegeben (so, wie man eben auch π schreibt anstelle von 3.141...). Wir weisen *Mathematica* an *numerisch* zu rechnen, indem wir das Argument mit einem Komma versehen (in *Mathematica* wird das Komma als Punkt eingegeben):

In[16]:=
```
Sin[1.]
```

Out[16]=
```
0.841471
```

Eine zweite Möglichkeit ist die Verwendung des Befehls `N[ ]`. Lassen wir uns zum Beispiel damit einen numerischen Wert für π

In[17]:=
```
N[Pi]
```

Out[17]=
```
3.14159
```

oder die Eulersche Zahl e ausgeben

In[18]:=
```
N[E]
```

Out[18]=
```
2.71828
```

Natürlich können wir auch eigene Funktionen definieren:

In[19]:=
```
f[x_] := x^2 + Sin[x] + a
```

Der Unterstrich in `x_` teilt *Mathematica* mit, dass `x` in diesem Ausdruck die unabhängige Variable (man spricht in *Mathematica* von einem "Pattern") ist. Die Verwendung von `:=` weist *Mathematica* an, die rechte Seite *jedes Mal neu auszuwerten,* wenn `f` aufgerufen wird. Daher haben wir hier auch kein `Out[...]` bekommen. Nun kann die neue Funktion `f` wie jede eingebaute Funktion verwendet werden (solange, bis Sie *Mathematica* beenden):

```
In[20]:= f[2]
Out[20]= 4 + a + Sin[2]

In[21]:= f[x]
Out[21]= a + x^2 + Sin[x]

In[22]:= x = 3; f[x]
Out[22]= 9 + a + Sin[3]
```

Achtung: man kann Funktionen auch nur mit einem = anstelle eines := definieren. Dann wird die rechte Seite *zuerst* ausgewertet (mit allen aktuellen Belegungen, ergibt also hier z.B. mit `x=3` den Wert `a + 9 + Sin[3]`): Dieser Wert wird dann (ein für alle Mal) als Funktionswert zugewiesen:

```
In[23]:= g[x_] = x^2 + Sin[x] + a
Out[23]= 9 + a + Sin[3]
```

`g` ist damit eine konstante Funktion, d.h., wir erhalten immer denselben Funktionswert, unabhängig vom Argument:

```
In[24]:= g[2]
Out[24]= 9 + a + Sin[3]
```

Es gibt also zwei Möglichkeiten: Funktionen von vornherein mit := definieren, oder sicherstellen, dass die Variable `x` nicht mit einem Wert belegt ist, also:

```
In[25]:= Clear[x]; g[x_] = x^2 + Sin[x] + a
Out[25]= a + x^2 + Sin[x]
```

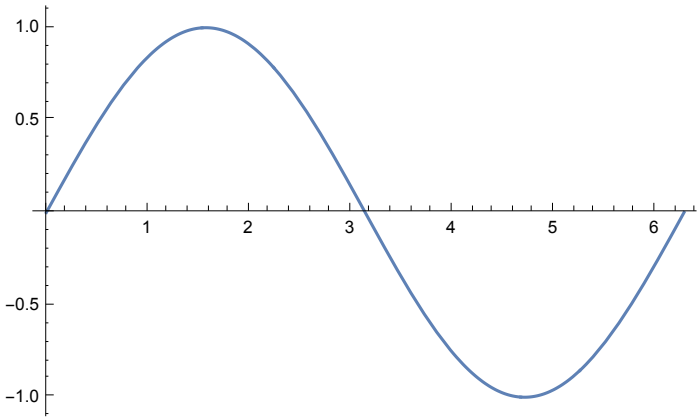
Mit **`Plot`** können Funktionen leicht gezeichnet werden:

Plot[*f*, {*x*, xmin, xmax}] zeichnet *f* als Funktion von *x* im Intervall von *xmin* bis *xmax*

Zeichnen wir zum Beispiel die Funktion `Sin[x]` im Intervall von 0 bis 2π:

```
In[26]:= Plot[Sin[x], {x, 0, 2 π}]
```

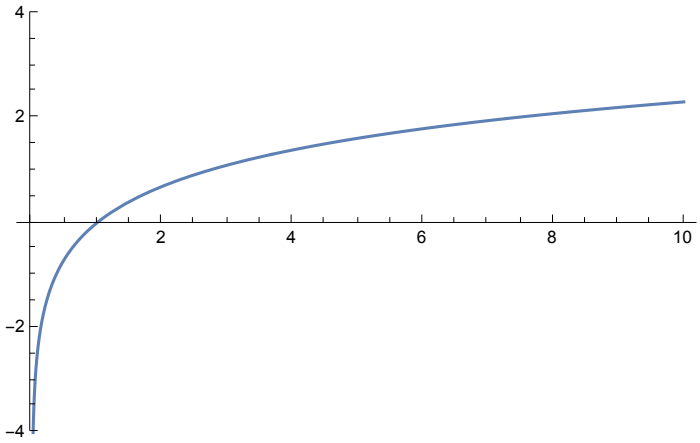
Out[26]=



Meist ist der von *Mathematica* dargestellte Ausschnitt der *y*-Achse passend. Man kann ihn aber auch selbst mit **`PlotRange`** festlegen. Wählen wir zum Beispiel für `Log[x]` das *y*-Intervall von -4 bis 4:

In[27]:=
```
Plot[Log[x], {x, 0, 10}, PlotRange → {-4, 4}]
```

Out[27]=



Mathematica enthält neben den eingebauten Funktionen auch eine Reihe von Standard-Zusatzpaketen (Algebra, Graphik, diskrete und numerische Mathematik, Zahlentheorie, Statistik, ...), die viele zusätzliche Funktionen bereithalten. Bei Bedarf wird das entsprechende Zusatzpaket geladen:

<< package`	liest das Paket `package` aus dem genannten Verzeichnis ein

Alternativ können Pakete auch mit **`Needs["package`"]`** geladen werden. (Achtung auf die Verwendung der richtigen Anführungszeichen!)

Übung: Zeichnen Sie die Funktion $y = \frac{1}{x-1}$ im x-Intervall von 0 bis 2 und im y-Intervall von -20 bis 20.

Gleichungen

Eine Gleichung wird in *Mathematica* mit einem doppelten Gleichheitszeichen eingegeben

In[28]:=
```
Sin[x]^2 + Cos[x]^2 == 1
```

Out[28]=
```
Cos[x]^2 + Sin[x]^2 == 1
```

Mit **`Simplify`** kann man versuchen die Gleichheit zu überprüfen:

In[29]:=
```
Simplify[%]
```

Out[29]=
```
True
```

Unsere Gleichung ist also richtig (True bzw. False sind die englischen Wörter für wahr bzw. falsch).

Die quadratische Gleichung

In[30]:=
```
gleichung = (x^2 - 2 x - 4 == 0);
```

kann mit dem Befehl **`Solve`** gelöst werden:

In[31]:=
```
loesung = Solve[gleichung, x]
```

Out[31]=
```
{{x → 1 - √5}, {x → 1 + √5}}
```

Mathematica liefert dabei die Lösung in Form von sogenannte **Ersetzungsregeln** $x \to$ *wert*. Der Vorteil dabei ist, dass man damit leichter weiter rechnen kann. Zum Beispiel können wir die beiden Lösungen in unsere Gleichung einsetzen und mit **`Simplify`** die Probe machen:

In[32]:=
```
Simplify[x^2 - 2 x - 4 /. loesung]
```
Out[32]=
```
{0, 0}
```

Mit dem **Ersetzungsoperator** /. wird durch *ausdruck /. x → wert* überall in *ausdruck* die Variable *x* durch *wert* ersetzt.

Falls es wie in unserem Fall mehrere Lösungen gibt, so kann man auf einzelne Lösungen mit dem Befehl

In[33]:=
```
loesung[[1]]
```
Out[33]=
$$\left\{x \to 1 - \sqrt{5}\right\}$$

zugreifen. Allgemein wird in *Mathematica* ein Ausdruck der Form

In[34]:=
```
{a, b, c, d, e};
```

als **Liste** bezeichnet. Man kann auf den n-ten Teil einer Liste mit `list[[n]]` zugreifen:

In[35]:=
```
%[[3]]
```
Out[35]=
```
c
```

Zusammenfassend gilt:

Solve[*a* == *b*, *x*]	löst die Gleichung $a == b$ mit x als Unbekannte
ausdruck /. loesung	setzt die *loesung* in *ausdruck* ein
loesung[[*n*]]	gibt den n'ten Eintrag der Liste *loesung* aus

Mathematica kann natürlich auch Systeme aus mehreren Gleichungen lösen, wie zum Beispiel:

In[36]:=
```
Solve[{x + y == a, x - y == 0}, {x, y}]
```
Out[36]=
$$\left\{\left\{x \to \frac{a}{2}, y \to \frac{a}{2}\right\}\right\}$$

Übung: Lösen Sie die Gleichung $x^2 - x - 12 = 0$.

Programme

Mathematica ist nicht nur ein Mathematikprogramm, sondern auch ein vollwertige Programmiersprache. Insbesondere stehen die üblichen Kontrollstrukturen und Schleifen zur Verfügung:

If[test, befehl1, befehl2]	Ist *test* wahr, führe *befehl1* aus, sonst *befehl2*

(*befehl2* ist optional)

Do[*j*, jmin, jmax, dj]	führt *befehl* mit $j = jmin,\ jmin + dj,\ ...,\ jmax$ aus
For[start, test, inkr, befehl]	führt einmal *start* aus und dann solange *befehl* und *inkr* bis *test* falsch ist
While[test, befehl]	führt *befehl* solange aus wie *test* wahr ist

(das Inkrement `dj` in der Do-Schleife ist optional mit Defaultwert `dj=1`).

Beispiel: Der Befehl PrimeQ überprüft, ob eine Zahl nur durch sich selbst oder eins teilbar, also eine Primzahl ist:

In[37]:=
```
PrimeQ[7]
```
Out[37]=
```
True
```

Mit diesem Befehl und einer Do-Schleife können wir eine Liste von Primzahlen kleiner gleich einer vorgegebenen Zahl erzeugen. Lassen wir uns zum Beispiel alle Primzahlen kleiner gleich 10 ausgeben:

In[38]:=
```
Do[
  If[PrimeQ[n], Print[n]],
  {n, 1, 10}];
```
```
2
3
5
7
```

Mit dem Befehl **`Module`** können mehrere Befehle übersichtlich zusammengefasst werden:

Module[{var1 = wert1, ...}, befehle]	Die *befehle* werden mit lokalen Werten für die aufgelisteten Variablen ausgeführt

Die einzelnen Befehle werden durch Strichpunkte getrennt. Das Ergebnis des letzten Befehls wird als Ergebnis des Blocks zurückgegeben.

Zum Beispiel können wir eine Funktion definieren, die die erste Primzahl ausgibt, die größer oder gleich einer vorgegebene Zahl ist:

In[39]:=
```
FindPrime[n_] := Module[{p = n},
  While[! PrimeQ[p], p++];
  p]
```

Dabei wird zuerst **`p`** mit **`n`** initialisiert. Dann wird **`p`** solange um eins erhöht (**`p++`** ist äquivalent zu **`p=p+1`**), wie der Primzahltest fehlschlägt (das Rufzeichen negiert den Test: aus wahr wird falsch und aus falsch wahr; hier wird also **`p`** um 1 erhöht, solange **`PrimeQ[p]`** falsch). Am Ende wird der gefundene Wert von **`p`** ausgegeben:

In[40]:=
```
FindPrime[1000]
```
Out[40]=
```
1009
```

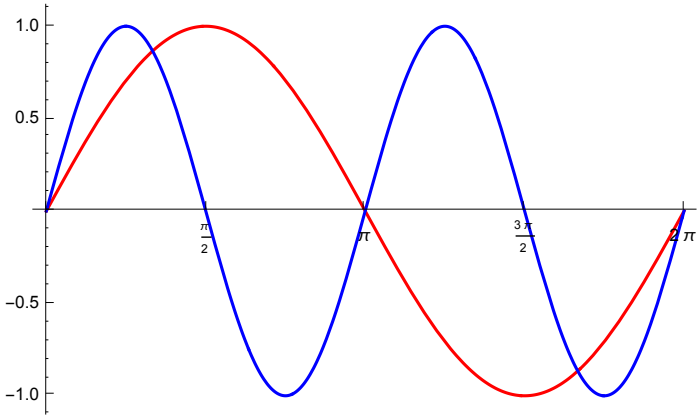
Übung: Verbessern Sie **`FindPrime`**, indem Sie berücksichtigen,dass Primzahlen (mit der Ausnahme von 2) immer ungerade sind (mit **`EvenQ[n]`** bzw. **`OddQ[n]`** können Sie testen, ob eine Zahl gerade oder ungerade ist).

Grafiken

Wir können auch mehrere Funktionen gleichzeitig darstellen:

In[41]:=
```
Plot[{Sin[x], Sin[2 x]}, {x, 0, 2 π}, Ticks → {{0, π/2, π, 3 π/2, 2 π}, Automatic},
 PlotStyle → {{RGBColor[1, 0, 0]}, {RGBColor[0, 0, 1]}}]
```

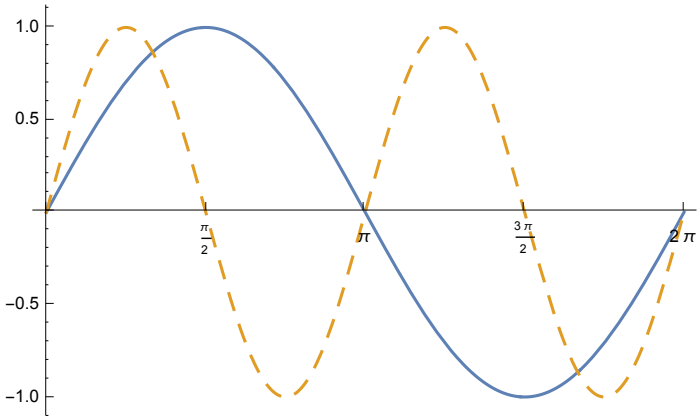
Out[41]=



Mit der Option **Ticks** können die Positionen der Achsenmarkierungen festgelegt werden und mit der Option **Plotstyle** können die einzelnen Funktionen in verschiedenen Farben dargestellt werden. Anstelle von Farben können auch verschiedene Linienformen verwendet werden:

In[42]:=
```
Plot[{Sin[x], Sin[2 x]}, {x, 0, 2 π}, Ticks → {{0, π/2, π, 3 π/2, 2 π}, Automatic},
 PlotStyle → {{}, Dashing[{0.025`, 0.025`}]}]
```

Out[42]=



Eine Fülle von weiteren Optionen finden Sie in der Online-Hilfe.

Falls Sie eine Grafik ins Web stellen oder in einem Textverarbeitungsprogramm verwenden wollen, so kann sie mit dem Befehl

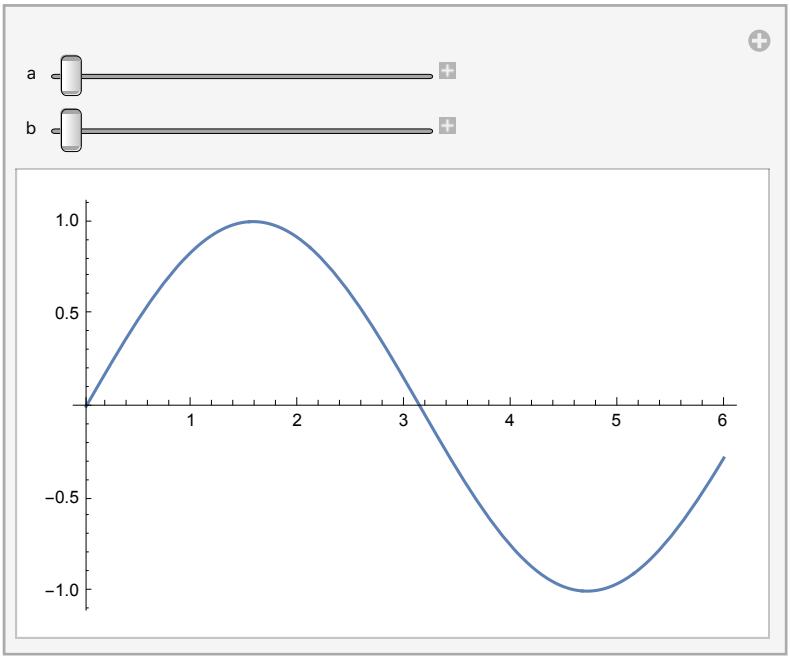
In[43]:=
```
Export["graph.eps", %, "EPS"];
```

auf die Festplatte geschrieben werden. Das Verzeichnis in dem die Datei landet können Sie mit dem Befehl **Directory[]** feststellen, bzw. mit **SetDirectory["pfad"]** ändern.

Mit dem Befehl **Manipulate[ausdruck, {a, amin, amax}]** kann ein Ausdruck für verschiedene Werte eines Parameters dargestellt werden. Der Parameter kann dann durch einen Schieberegler variiert werden und die Änderungen werden sofort veranschaulicht.

In[44]:=
```
Manipulate[Plot[Sin[a x + b], {x, 0, 6}], {a, 1, 4}, {b, 0, 10}]
```

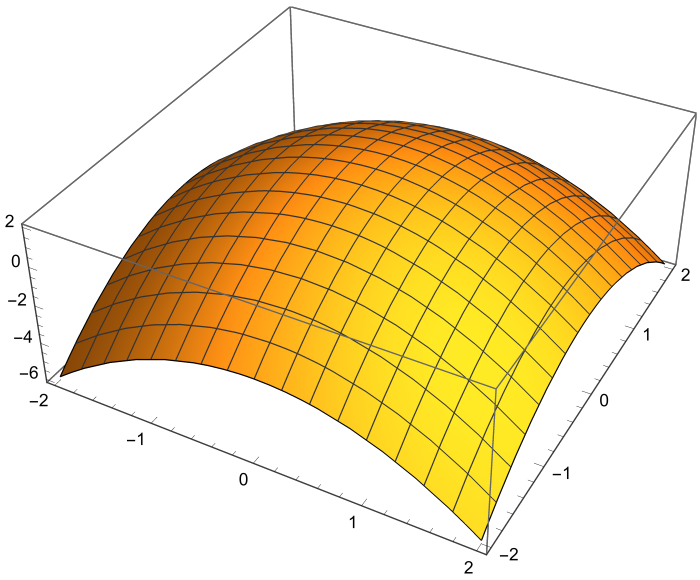
Out[44]=



Zur Veranschaulichung einer reellwertigen Funktion von zwei Variablen kann der Befehl **Plot3D[**f[x, y], {x, xmin, xmax}, {y, ymin, ymax}**]** verwendet werden.

In[45]:=
```
Plot3D[2 - x^2 - y^2, {x, -2, 2}, {y, -2, 2}]
```

Out[45]=



Was geht noch?

Natürlich beherrscht *Mathematica* alle gängigen mathematischen Funktionen und Operationen. Hier möchte ich Ihnen nur noch einen kleinen Ausblick in die grafischen Fähigkeiten von *Mathematica* geben.

Vielleicht haben Sie schon mal Bilder der Mandelbrotmenge, die als klassisches Beispiel einer

fraktalen Menge gilt, gesehen. Um die Erzeugung dieser Bilder zu verstehen brauchen Sie nur zu wissen, was eine komplexe Zahl ist (wenn Sie das nicht wissen, erfreuen
Sie sich einfach an den Bildern;-). Für eine gegebene komplexe Zahl c werden rekursiv die Zahlen

$$z_0 = 0, \quad z_1 = z_0^2 + c, \quad z_2 = z_1^2 + c, \quad \ldots$$

berechnet und die Anzahl der Schritte k notiert, nach denen der Absolutbetrag von z_k größer als zwei ist. Da das unter Umständen nie passiert, wird nach einer festen Anzahl von Schritten, z.B. 60, abgebrochen.

Folgende Funktion berechnet die Anzahl der notwendigen Schritte:

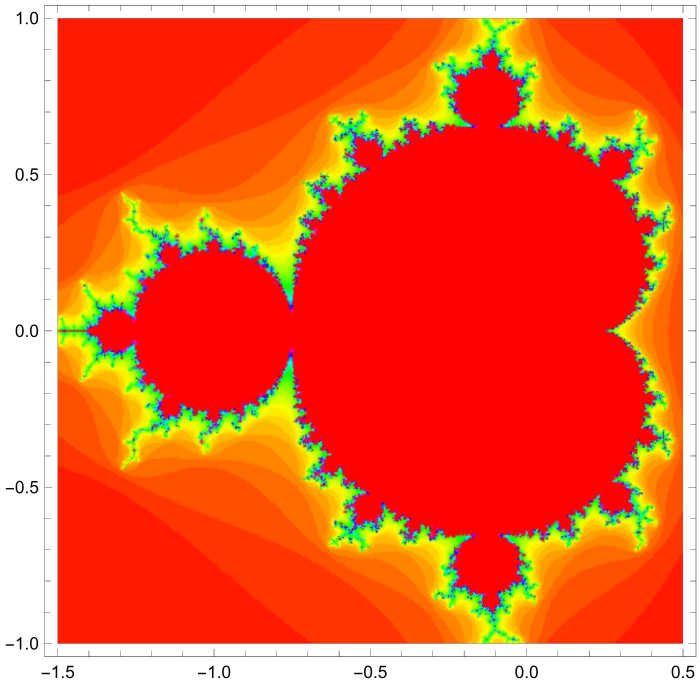
In[46]:=
```
Mandel = Compile[{{c, _Complex}},
   Block[{z = 0. + 0. i, k = 0},
    While[Abs[z] < 2 && k < 60, z = z^2 + c; k = k + 1];
    k]
  ];
```

Das doppelte & entspricht einer logischen Und-Verknüpfung; der Test ist also genau dann wahr, wenn beide Teile wahr sind. Außerdem wurde der ganze Block unter ein `Compile[...]` gesteckt. Das ist zwar nicht notwendig, es veranlasst *Mathematica* aber, die Funktion, unter der Annahme dass das Argument c komplex ist, zu compilieren, was einen erheblichen Geschwindigkeitsvorteil bringt. Ausgegeben wird die Anzahl der durchgeführten Schritte k.

Führt man diese Berechnung für verschiedene komplexe Werte `c=x + I y` durch und veranschaulicht das Ergebnis, indem man den Punkt mit den Koordinaten (x, y) entsprechend der Anzahl der zugehörigen Schritte einfärbt, so erhält man die zuvor erwähnten Bilder der Mandelbrotmenge. In *Mathematica* kann das mit folgendem Befehl geschehen:

In[47]:=
```
DensityPlot[Mandel[x + I y], {x, -1.5, 0.5}, {y, -1, 1},
 ColorFunction → Hue, Mesh → False, PlotPoints → 20, MaxRecursion → 5]
```

Out[47]=



Hier werden Werte zwischen $-1.5 \le x \le 0.5$ und $-1 \le y \le 1$ berechnet. Die Option **`ColorFunction -> Hue`** bringt etwas Farbe ins Bild (ansonsten würden den Funktionswerten nur Graustufen zugeordnet), die Option **`Mesh -> False`** verhindert, dass ein langweiliges Koordinatengitter über unser schönes Bild gezeichnet wird, und **`PlotPoints -> 400`** legt fest, wie viele Punkte berechnet werden sollen (der Defaultwert ist 25, womit das Bild noch etwas kantig aussieht).

Das mathematisch interessante an der Mandelbrotmenge (der rote Fleck in der Mitte) ist ihr fraktaler Rand, der ziemlich unregelmäßig und alles andere als glatt ist. Er wird auch nicht glätter, wenn Sie einen Ausschnitt davon vergrößern: egal, wie nahe Sie heranzoomen, der Rand bleibt immer gleich unregelmäßig, und es wiederholen sich immer wieder die gleichen Strukturen. Das ist eine Eigenschaft, die charakteristisch für fraktale Mengen ist und als "selbstähnlich" bezeichnet wird.

Übung: Zeichnen Sie den Bereich $0.365 \le x \le 0.375$ und $0.095 \le y \le 0.1059$.