Einführung in *Mathematica* - Teil 2 -

zur Vorlesung *Mathematische Methoden der Physik* im WiSe 2011/12

Norbert Dragon und Michael Flohr (mit Unterstützung von *Martin Paech*)
16. 12. 2011

Analysis

Differentialgleichungen

Analytische Integration (Man beachte die Integrationskonstante `C[1]`.)

```
DSolve[y'[x] - x y[x] == 0, y[x], x]
```

$$\left\{\left\{y[x] \to e^{\frac{x^2}{2}}\, C[1]\right\}\right\}$$

Beim Zuweisen zu einer neuen Funktion `z` ist wiederum der Ersetzungsoperator notwendig; `[[1]]` "löst" sozusagen die verschachtelnde Liste auf.

```
z[x_] = y[x] /. DSolve[y'[x] - x y[x] == 0, y[x], x][[1]]
```

$$e^{\frac{x^2}{2}}\, C[1]$$

Nun kann, bis auf die verbleibende Konstante, mit der neuen Funktion numerisch gerechnet werden ...

```
z[1]
```

$$\sqrt{e}\, C[1]$$

```
N[%]
```

$$1.64872\, C[1]$$

Die Eleminierung der Konstanten erfolgt, wie gehabt, über eine Ersetzung mit einer Transformationsregel (`I` ist die komplexe Zahl):

```
% /. C[1] → I
```

$$0. + 1.64872\, i$$

Analytische Integration mit Randbedingung:

```
DSolve[{y''[x] + ω^2 y[x] == 0, y[0] == 1, y'[0] == 1 / 2}, y[x], x]
```

$$\left\{\left\{y[x] \to \frac{2\,\omega\,\text{Cos}[x\,\omega] + \text{Sin}[x\,\omega]}{2\,\omega}\right\}\right\}$$

Allgemeine Lösung einer Differentialgleichung in Form der sog. “reinen” Funktion (beachte, dass das Argument bei `y` nicht mit angegeben wird):

```
DSolve[y''[x] + ω^2 y[x] == 0, y, x]
```

```
{{y → Function[{x}, C[1] Cos[x ω] + C[2] Sin[x ω]]}}
```

Lösung für einen bestimmten Fall durch Substitution, vgl. erstes Beispiel:

```
(y[x] /. %)[[1]]
```

```
C[1] Cos[x ω] + C[2] Sin[x ω]
```

Hinweis: Die Substitutionsanweisung schreibt man vorteilhafterweise in runde Klammern, bevor die Liste aufgelöst wird, denn *Mathematica* ist sehr logisch: Geschweifte Klammern erzeugten gleich wieder eine neue Liste, für die eine weitere Operation mit `[[1]]` nötig wäre.

```
{y[x] /. %%}[[1]][[1]]
```

```
C[1] Cos[x ω] + C[2] Sin[x ω]
```

Vektoranalysis

```
Clear[x, y, z, f, g, h]
```

Laden eines Paketes für erweitere Funktionen - das kann einen Augenblick dauern ...

```
<< VectorAnalysis`
```

Hinweis: Zuerst sollte man das Koordinatensystems und seine Variablen festlegen (*Mathematica* gibt zwar `Cartesian`, d.h. “kartesisch”, aber konsistent zu seinen Groß- und Kleinschreibungsregeln `Xx`,`Yy` und `Zz` vor. Ein anderes mögliches Koordinatensystem ist z. B. `Spherical` mit den Variablen `Rr`, `Ttheta`, `Pphi`.)

```
SetCoordinates[Cartesian[x, y, z]]
```

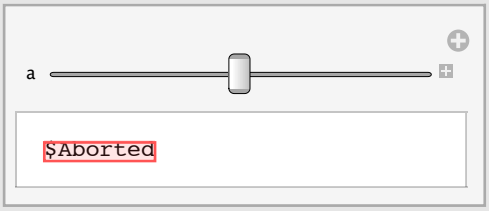
```
Cartesian[x, y, z]
```

Definition einer(skalaren) Funktion:

```
f[x_, y_, z_] := x^2 (3 - 2 y^2) - 2 (x^2 + y^2) z^2 + 3 (y^2 + z^2)
```

SIe haben schon Äquipotentiallinien in einer Hausübung zeichen müssen. Hier plotten wir eine Äquipotentialfläche:

```
Manipulate[
 ContourPlot3D[f[x, y, z] == a, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}], {{a, 3}, 1, 5}]
```



Berechnung des Gradienten von `f` im oben vorgegebenen Koordinatensystem:

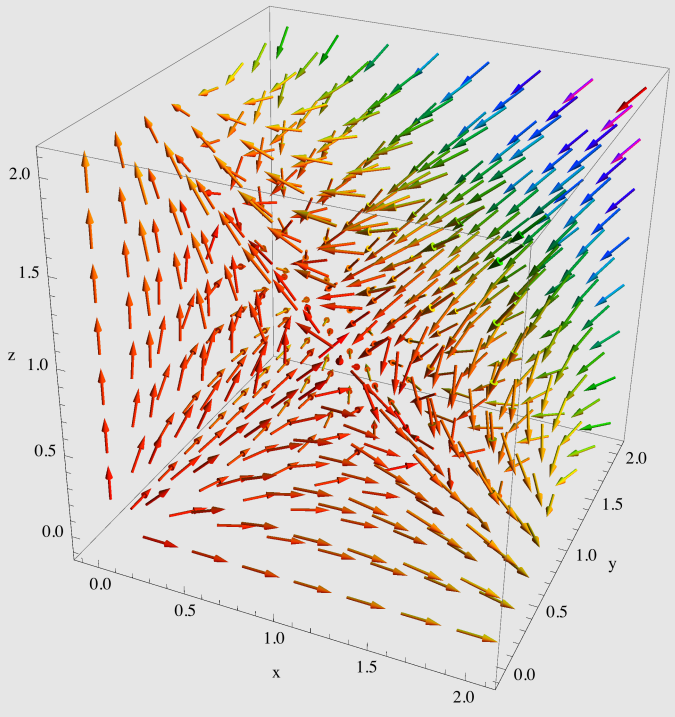
```
g = Grad[f[x, y, z]]
```

$$\left\{2\,x\,\left(3-2\,y^2\right)-4\,x\,z^2,\ 6\,y-4\,x^2\,y-4\,y\,z^2,\ 6\,z-4\,\left(x^2+y^2\right)\,z\right\}$$

Achtung: `g` ist keine Funktion gemäß der für `f` gebrauchten Definition, sondern eine “vektorartige” Liste!

Darstellung eines dreidimensionalen Vektorfeldes: Unter den benutzten Darstellungsoptionen sei die universelle, auch in `Plot` verwendbare `AxesLabel` hervorgehoben.

```
VectorPlot3D[g, {x, 0, 2}, {y, 0, 2}, {z, 0, 2},
 AxesLabel → {"x", "y", "z"}, VectorStyle → "Arrow3D",
 VectorScale → {Small, Small, None}, VectorColorFunction → Hue]
```



Wir berechnen nun die zwei grundlegenden Differentationen für Vektorfelder, die Divergenz und die Rotation. Letztere sollte für ein Gradientenfeld natürlich verschwinden:

```
h = Div[g]
r = Curl[g]
```

$$12 - 4\,x^2 + 2\,\left(3 - 2\,y^2\right) - 4\,\left(x^2 + y^2\right) - 8\,z^2$$

$$\{0, 0, 0\}$$

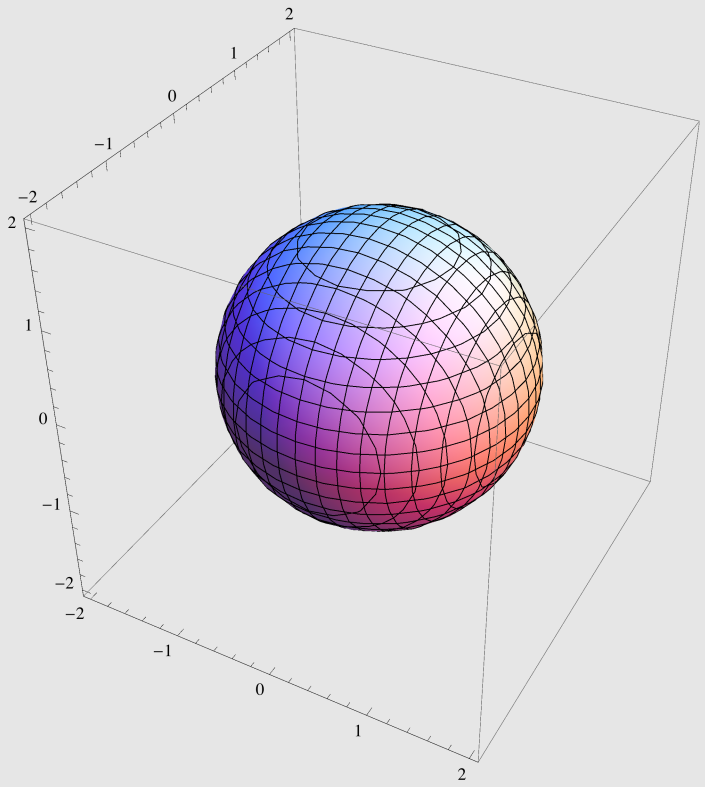
```
FullSimplify[%%]
```

$$-2\,\left(-9 + 4\,x^2 + 4\,y^2 + 4\,z^2\right)$$

Achtung: `h` ist ebenfalls keine Funktion gemäß der für `f` gebrauchten Definition, sondern eine "skalarartige" (also einelementige) Liste!

Wie man nach der Vereinfachung mittels `FullSimplify` sieht, ist diese Funktion rotationssymmetrisch:

```
ContourPlot3D[h == 0, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}]
```



Listen

Listenerzeugung

Ein einfaches Beispiel zum Einstieg:

```
Table[n, {n, 0, 16}]
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
```

```
Table[n, {n, 0, 16, 2}]
```

```
{0, 2, 4, 6, 8, 10, 12, 14, 16}
```

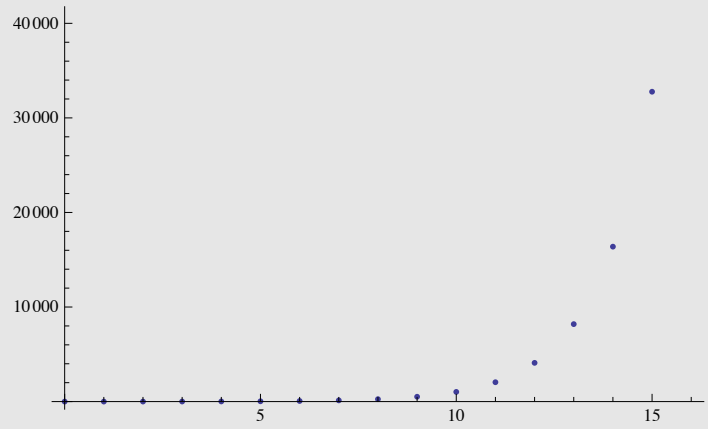
Table kann auch eine Liste von Listen generieren, beispielsweise Wertetabellen:

```
Table[{n, 2^n}, {n, 0, 16}]
```

```
{{0, 1}, {1, 2}, {2, 4}, {3, 8}, {4, 16}, {5, 32}, {6, 64}, {7, 128}, {8, 256}, {9, 512},
 {10, 1024}, {11, 2048}, {12, 4096}, {13, 8192}, {14, 16 384}, {15, 32 768}, {16, 65 536}}
```

Listen dieser Art mit numerischen Daten lassen sich unkompliziert graphisch darstellen:

```
ListPlot[%]
```



Symbolisch geht's mit der Listenerzeugung freilich auch:

```
Table[(a + b)^n, {n, 0, 4}]
```

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

```
liste = Expand[%]
```

$$\left\{1, a+b, a^2+2\,a\,b+b^2, a^3+3\,a^2\,b+3\,a\,b^2+b^3, a^4+4\,a^3\,b+6\,a^2\,b^2+4\,a\,b^3+b^4\right\}$$

Listenmanipulationen

Zunächst wird beispielhaft das erste Listenelement der vorletzten Programmausgabe entfernt ...

```
Drop[%%, 1]
```

$$\left\{a+b, (a+b)^2, (a+b)^3, (a+b)^4\right\}$$

... und dann das erste "extrahiert":

```
First[%]
```

$$a + b$$

Den letzten Listeneintrag erhält man "natürlich" mit **Last**!

```
Last[liste]
```

$$a^4 + 4\,a^3\,b + 6\,a^2\,b^2 + 4\,a\,b^3 + b^4$$

Ausschneiden von Listenteilen erlaubt **Part**.

```
Part[liste, 2 ;; 3]
```

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

Ein alternativer Weg geht mittels des uns schon bekannten **[[...]]**:

```
liste[[2 ;; 3]]
```

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

Der Zugriff auf ein Element erfolgt in gewohnter Manier:

```
%[[1]]
```

$$a + b$$

```
Table[{n, 1/n}, {n, 1, 7}]
```

$$\left\{\{1, 1\}, \left\{2, \frac{1}{2}\right\}, \left\{3, \frac{1}{3}\right\}, \left\{4, \frac{1}{4}\right\}, \left\{5, \frac{1}{5}\right\}, \left\{6, \frac{1}{6}\right\}, \left\{7, \frac{1}{7}\right\}\right\}$$

Flatten reduziert die beiden Ebenen zuvor generierter Liste auf eine. Dies ist ein wichter Operator in *Mathematica*.

```
Flatten[%]
```

$$\left\{1, 1, 2, \frac{1}{2}, 3, \frac{1}{3}, 4, \frac{1}{4}, 5, \frac{1}{5}, 6, \frac{1}{6}, 7, \frac{1}{7}\right\}$$

Die Summe aller Listenelemente läßt sich mit **Total** berechnen.

```
Total[%]
```

$$\frac{4283}{140}$$

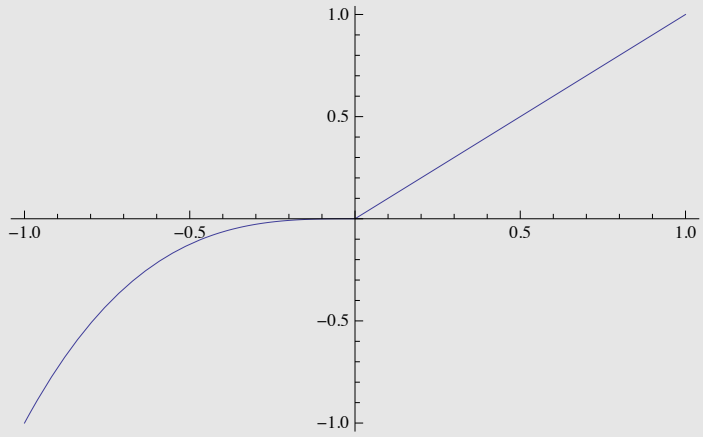
Bedingungen

Abschnittsweise definierte Funktionen

Eine reelle Funktion, die auf zwei Intervallen unterschiedlich definiert ist,

```
Clear[f, g]
f[x_] := If[x < 0, x^3, x]
```

```
Plot[f[x], {x, -1, 1}]
```



sowie deren bestimmtes Integral,

```
Integrate[f[x], {x, -1, 1}]
```

$$\frac{1}{4}$$

deren "Stammfunktion"

```
Integrate[f[x], x]
```

$$\begin{cases} \frac{x^4}{4} & x \leq 0 \\ \frac{x^2}{2} & \text{True} \end{cases}$$

und deren erste Ableitung

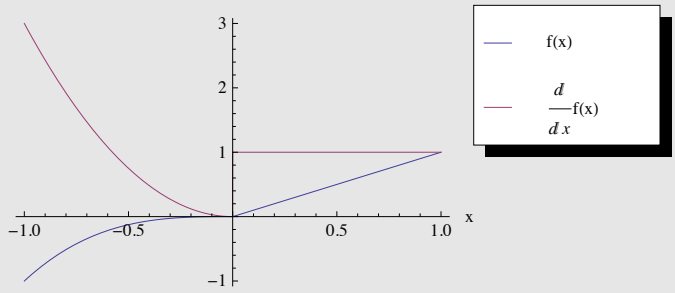
```
g[x_] = D[f[x], x]
```

```
If[x < 0, 3 x^2, 1]
```

Das Paket **PlotLegends** muß zum Erzeugen einer Legende geladen werden.

```
<< PlotLegends`
```

```
Plot[{f[x], g[x]}, {x, -1, 1}, AxesLabel → {"x"},
 PlotLegend → {"f(x)", "d/dx f(x)"}, LegendPosition → {1, 0}]
```



Weitere Anwendungen

```
primzahlen = Table[Prime[i], {i, 20}]
```

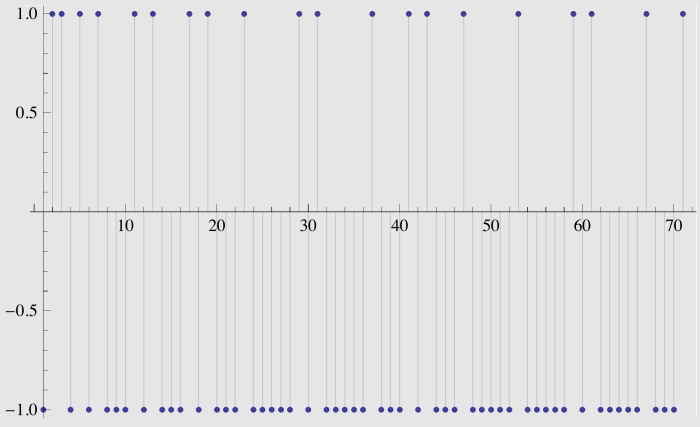
```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71}
```

```
j = Last[primzahlen]
```

```
71
```

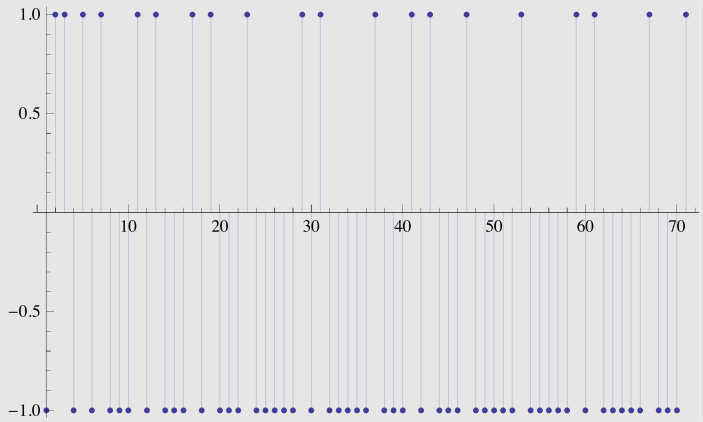
MemberQ durchsucht die Liste, ob ein bestimmtes Element enthalten ist.

```
DiscretePlot[If[MemberQ[primzahlen, i], 1, -1], {i, 1, j}]
```



Durchsuchen ist aber eigentlich gar nicht nötig, da die Liste aufsteigend sortiert ist und aufsteigend "abgearbeitet" wird.

```
DiscretePlot[
 If[i == First[primzahlen], primzahlen = Drop[primzahlen, 1]; 1, -1], {i, 1, j}]
```



Nach obiger "Abarbeitung" der Primzahlenliste ist diese leer.

```
primzahlen
```

```
{}
```

Logische Ausdrücke

Alle Vergleiche können nur zwei Werte annehmen: Wahr oder falsch.

```
2^0 == 1
```

```
True
```

```
2 < 0
```

```
False
```

- **Verknüpfung zweier Vergleiche mit "und" bzw. "oder"**

```
3 > E && 3 < Pi
```

```
True
```

Äquivalente Eingabe:

3 > E ⋀ 3 < Pi

```
3 < E || 3 < Pi
```

```
True
```

Äquivalente Eingabe:

```
3 > E ⋁ 3 > Pi
```

Negation

```
! (a > b)
```

```
a ≤ b
```

Verschachtelte `If`-Befehle

```
Integrate[If[x < 0, If[x == -1, 1, 2], 3], x]
```

$$\begin{cases} 2\,x & x \le -1 \,||\, -1 < x \le 0 \\ 3\,x & \text{True} \end{cases}$$

Schleifenstrukturen und Iteration

```
i = 0;
While[i < 3, Print[i]; i = i + 1]
```

```
0

1

2
```

Eine klassische "`for`"-Schleife, wie sie wohl nahezu jede (imperative) Programmiersprache kennt:

```
For[i = 0, i < 3, i = i + 1, Print[i]]
```

```
0

1

2
```

In *Mathematica* gibt es auch das einfachere (und meist schnellere), zu **`For`** äquivalente **`Do`**,

```
Do[Print[i], {i, 0, 2}]
```

```
0

1

2
```

welches von der Struktur beispielsweise analog zu **`Table`** ist.

```
Table[i, {i, 0, 2}]
```

```
{0, 1, 2}
```

Anwendungen

```
For[i = 0, i < 3, i = i + 1, k = i]
```

```
k
```

```
2
```

Eine (Summations)schleife in mathematisch etwas vertrauterer Darstellung

```
k = Sum[1, {i, 1, 10}]
```

```
10
```

Dieselbe sieht "programmiert" wie folgt aus:

```
k = 0;
For[i = 1, i ≤ 10, i = i + 1, k = k + 1]
k
```

```
10
```

Ein weiteres Beispiel

```
k = ∑_{i=1}^{10} i
```

```
55
```

```
k = 0;
For[i = 1, i ≤ 10, i = i + 1, k = k + i]
k
```

```
55
```

Fakultät, iterativ

```
Clear[fi, fr]
```

```
fi[n_] := Product[m, {m, 2, n}]
```

```
fi[137]
```

```
5 012 888 748 274 991 661 034 926 292 112 253 883 237 205 694 398 754 483 388 962 668 892 510 972 ↘
  746 226 260 034 675 717 797 072 343 372 830 591 567 227 826 571 884 373 881 355 612 819 314 826 ↘
  377 917 827 129 740 056 802 397 016 509 378 163 883 274 055 583 382 110 208 000 000 000 000 000 ↘
  000 000 000 000 000 000
```

Kontrollrechung: Haben wir das richtig programmiert?

```
137 !
```

```
5 012 888 748 274 991 661 034 926 292 112 253 883 237 205 694 398 754 483 388 962 668 892 510 972 ↘
  746 226 260 034 675 717 797 072 343 372 830 591 567 227 826 571 884 373 881 355 612 819 314 826 ↘
  377 917 827 129 740 056 802 397 016 509 378 163 883 274 055 583 382 110 208 000 000 000 000 000 ↘
  000 000 000 000 000 000
```

Rekursion

Fakultät, rekursiv

```
fr[n_] := n fr[n - 1]
```

```
fr[1] = 1;
```

```
fr[137]
```

```
5 012 888 748 274 991 661 034 926 292 112 253 883 237 205 694 398 754 483 388 962 668 892 510 972 ↘
  746 226 260 034 675 717 797 072 343 372 830 591 567 227 826 571 884 373 881 355 612 819 314 826 ↘
  377 917 827 129 740 056 802 397 016 509 378 163 883 274 055 583 382 110 208 000 000 000 000 000 ↘
  000 000 000 000 000 000
```

Die Funktion **Timing** gibt die Verarbeitungszeit in Sekunden an.

```
Timing[fr[10 000];]
```

```
$RecursionLimit::reclim : Recursion depth of 256 exceeded. ≫
```

```
{0.001318, Null}
```

Um eine Systemvariable wie **$RecursionLimit** nur in einer Eingabezeile zu verändern, muß die beeinflußte Funktion in **Block** verschachtelt werden.

```
Timing[Block[{$RecursionLimit = 10 004}, rf[10 000];]]
```

```
{0.000019, Null}
```

Geschwindigkeitsvergleich. Die eingebauten Funktionen von *Mathematica* sind hochoptimiert und meist unschlagbar schnell. Dass die rekursive Definition hier gewinnt, liegt daran, dass *Mathematica* einmal berechnete Werte cached, und das die eingebaute Funktion für die Fakultät wesentlich mehr kann. Dahinter steckt nämlich die Γ-Funktion, die analytische Fortsetzung der Fakultät auf die ganzen komplexen Zahlen.

```
Timing[fi[10 000];]
```

```
{0.00765, Null}
```

```
Timing[10 000 !;]
```

```
{0.000952, Null}
```