Dieses Notebook ist ein sehr praktischer Ersatz für den Taschenrechner. Das letzte Ergebnis wird in einer nachfolgenden Zeile ausgegeben.

Sie können Diese Software kostenlos installieren uns selbst probieren. Hier ist die Prozedur:

I. Installieren Sie die Enthough Python Distribution (EPD) von http://www.enthought.com/products/edudownload.php (Als Student bekommen Sie diese kostenlos, aber auch die EPDfree Version ist völlig ausreichend)
II. Öffnen Sie ein Kommandozeilenfenster, wechseln Sie in ein von Ihnen definiertes Arbeitsverzeichnis und geben Sie den Befehl folgenden Befehl ein: 'ipython notebook --pylab inline'.
(Wenn Sie vorhaben, das öfter zu machen, können Sie sich dafür natürlich auch einen Link, oder eine Batch-Datei generieren. Beachten Sie auch, dass animierte Plots zur Zeit nicht mit der 'inline'-Option kompatibel sind.)
III. Klicken Sie 'New Notebook', und schon können Sie losrechnen

Beachten Sie bitte, dass dieses Notebook zur Zeit nicht IE 9 unterstützt. Aber Firefox oder Crome funktionieren sehr gut.

Um eine neue Pogramm-Zelle zu öffnen, klicken Sie einfach auf 'Below' Oder drücken Ctrl-m b.

Um darin Text, so wie diesen zu schreiben, oder gar mathematische Formeln, wie, z.B. diese Definition der 1-dimensionalen Fouriertransformation

$$FT[f(x)] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-2\pi i x k} \, dk$$

mit LaTeX-Befehlen müssen Sie als Format 'Markdown' einstellen.

Hier, z.B. die Berechnung der Wurzel aus 2 (zum Ausführen Shift+Enter):

In [1]:
```
sqrt(2)
```

Out[1]:
```
1.4142135623730951
```

Es gibt eine Reihe von weiteren eingebauten Mathematikfunktionen. Normalerweise müssen diese mit dem Befehl 'from numpy import *' eingebunden werden. Aber wenn Sie dieses Notenook mit der '--pylab' Option gestartet haben, dann können diese Funktionen direkt verwendet werden.

Hier z.B. der natürliche Logarithmus von 2 multipliziert mit 2, d.h. sqrt(2)*ln(2):

In [2]:
```
sqrt(2)*log(2)
```

Out[2]:
```
0.98025814346854723
```

In dem obigen Beispiel wird nur das letzte Ergebnis angezeigt. Es können aber jederzeit Zwischenergebnisse ausgegeben werden, sogar schön formatiert:

In [3]:
```
print "sqrt(2) = ",sqrt(2)
print "ln(2) = ",log(2)
```

```
sqrt(2) =  1.41421356237
ln(2) =  0.69314718056
```

Natprlich kann man auch Variablen definieren und mit diesen rechnen:

In [4]:
```
x = 2
print "x = ",x
```

```
x =  2
```

Es sind auch einige Konstanten definiert, wie z.B. pi oder die Eulersche Zahl. Versuchen Sie daher, diese Namen besonderer Konstanten nicht mit Variablen zu belegen.

In [5]:
```
y = pi
print "y = ",y
print "y/pi = ",y/pi
eul = e
print "Eulersche Zahl = ",e," => ln(e) = ",log(e)
```

```
y =  3.14159265359
y/pi =  1.0
Eulersche Zahl =  2.71828182846  => ln(e) =  1.0
```

Wir können auch ganz einfach von Grad in Radiant umrechnen:

In [6]:
```
deg2rad = pi/180
print "1 deg = ",deg2rad," rad = ",1000*deg2rad," mrad"
print "1 rad = ",1/deg2rad," deg"
```

```
1 deg =  0.0174532925199  rad =  17.4532925199  mrad
1 rad =  57.2957795131  deg
```

Natürlich kann man auch Funktionen an mehreren Positionen simultan berechnen. Hier, z.B. die Zahlen 1 .. 5 und deren Inverses:

In [7]:
```
x = frange(1.0,5.0)
y = 1/x
print x
print y
```

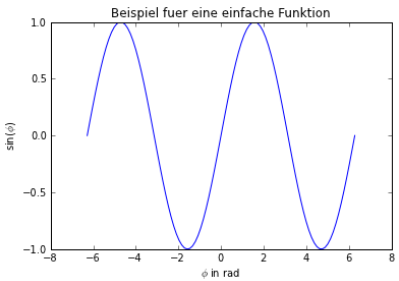
```
[ 1.  2.  3.  4.  5.]
[ 1.          0.5         0.33333333  0.25        0.2       ]
```

Man kann diese Funktionen auch sehr einfach graphisch darstellen. Dazu sollten Sie sich zunächst eine die Werte auf der Abszisse (oder x-Achse) generrieren, und dann die Werte Ihrer Funktion berechnen.

In diesem einfachen Beispiel ist die Abszisse der Winkel ϕ und die Funktion einfach $\sin(\phi)$.

In [8]:
```
phi = linspace(-2*pi,2*pi,100) # 100 phi-Werte zwischen -2pi und 2pi
f = sin(phi)
plot(phi,f)
# Um den Plot schöner zu machen, beschriften wir noch die Achsen:
xlabel('$\phi$ in rad')
ylabel('sin($\phi$)')
# Bitte vermeiden Sie die Verwendung von Umlauten (außer in Kommentaren):
title('Beispiel fuer eine einfache Funktion')
```

Out[8]:
```
<matplotlib.text.Text at 0x51372f0>
```



IPython lässt sich auch sehr gut verwenden, um Tabellen zu bearbeiten. Als Beispiel verwenden wir Aufgabe 4 aus dem ersten Übungsblatt. Dort war die Frage danach, welche von 5 Uhren die beste ist.

Dazu müssen wir einfach nur nach dem Zeitunterschied von Tag zu Tag schauen und feststellen, welche Uhr am konstantesten läuft.

Bevor wir aber losrechnen können, müssen wir zunächst die Tabelle aus dem Übungsblatt in eine für IPython verständliche Form bringen. Das machen wir, indem wir einen String definieren, dessen Inhalt wir mittels Copy-Paste aus der PDF-Datei holen und die Buchstaben am Anfang der Zeilen entfernen.

Wenn wir diesen String ersteinmal definiert haben, dann müssen wir nur noch aus den 'hh:mm:ss' Angaben ganze Sekunden machen. Dazu konvertieren wir das 'hh:mm:ss' in die Formel

$$t = ((hh * 60) + mm * 60) + ss$$

indem wir einfach die ':'-Zeichen durch die entsprechenden Faktoren ersetzen.

In [9]:
```
timeStr = """12:36:40 12:36:56 12:37:12 12:37:27 12:37:44 12:37:59 12:38:14
11:59:59 12:00:02 11:59:57 12:00:07 12:00:02 11:59:56 12:00:03
15:50:45 15:50:45 15:52:41 15:53:39 15:54:37 15:55:35 15:56:33
12:03:59 12:02:52 12:01:45 12:00:38 11:59:31 11:58:24 11:57:17
12:03:59 12:02:49 12:01:54 12:01:52 12:01:32 12:01:22 12:01:12"""
print "Originaltext:", timeStr
timeStr2 = "Uhrzeit_in_sec = array((((("+timeStr.replace('\n',',\n((').replace(':',')*60+')\
.replace(' ',', ((')+")).reshape(5,7)\n"
print "\nNach der Konvertierung in Sekunden und in der korrekten Formatierung als Matrix:\n\
(die folgenden 5 Zeilen einfach nur in ein neues Code-Fenster kopieren)\n",timeStr2,"\n"
```

```
Originaltext: 12:36:40 12:36:56 12:37:12 12:37:27 12:37:44 12:37:59 12:38:14
11:59:59 12:00:02 11:59:57 12:00:07 12:00:02 11:59:56 12:00:03
15:50:45 15:50:45 15:52:41 15:53:39 15:54:37 15:55:35 15:56:33
12:03:59 12:02:52 12:01:45 12:00:38 11:59:31 11:58:24 11:57:17
12:03:59 12:02:49 12:01:54 12:01:52 12:01:32 12:01:22 12:01:12

Nach der Konvertierung in Sekunden und in der korrekten Formatierung als Matrix:
(die folgenden 5 Zeilen einfach nur in ein neues Code-Fenster kopieren)
Uhrzeit_in_sec = array((((12)*60+36)*60+40, ((12)*60+36)*60+56, ((12)*60+37)*60+12, ((12)*60+37)*60+27, ((12)*60+37)*60+44, ((12)*60+37)*60+59, ((12)*60+38)*60+14,
((11)*60+59)*60+59, ((12)*60+00)*60+02, ((11)*60+59)*60+57, ((12)*60+00)*60+07, ((12)*60+00)*60+02, ((11)*60+59)*60+56, ((12)*60+00)*60+03,
((15)*60+50)*60+45, ((15)*60+50)*60+45, ((15)*60+52)*60+41, ((15)*60+53)*60+39, ((15)*60+54)*60+37, ((15)*60+55)*60+35, ((15)*60+56)*60+33,
((12)*60+03)*60+59, ((12)*60+02)*60+52, ((12)*60+01)*60+45, ((12)*60+00)*60+38, ((11)*60+59)*60+31, ((11)*60+58)*60+24, ((11)*60+57)*60+17,
((12)*60+03)*60+59, ((12)*60+02)*60+49, ((12)*60+01)*60+54, ((12)*60+01)*60+52, ((12)*60+01)*60+32, ((12)*60+01)*60+22, ((12)*60+01)*60+12)).reshape(5,7)
```

In einem zweiten Schritt müssen wir nur noch die obige Tabelle in ein neues Code-Fenster kopieren und die Differenzen von Tag zu Tag für die einzelnen Uhren berechnen.

Zur Auswertung schauen wir uns die Varianz der Sekunden für einen Tag und den Mittelwert an. Anhand dieser Information können wir dann ganz leicht entscheiden, welche Uhr die beste (z.B. im Sinne von tauglich für die Bestimmung des Längengrades zur Navigation aus See) ist.

In [10]:
```
Uhrzeit_in_sec = array((((12)*60+36)*60+40, ((12)*60+36)*60+56, ((12)*60+37)*60+12, ((12)*60+37)*60+27, ((12)*60+37)*60+44, ((12)*60+37)*60+59, ((12)*60+38)*60+14,
 ((11)*60+59)*60+59, ((12)*60+00)*60+02, ((11)*60+59)*60+57, ((12)*60+00)*60+07, ((12)*60+00)*60+02, ((11)*60+59)*60+56, ((12)*60+00)*60+03,
 ((15)*60+50)*60+45, ((15)*60+50)*60+45, ((15)*60+52)*60+41, ((15)*60+53)*60+39, ((15)*60+54)*60+37, ((15)*60+55)*60+35, ((15)*60+56)*60+33,
 ((12)*60+03)*60+59, ((12)*60+02)*60+52, ((12)*60+01)*60+45, ((12)*60+00)*60+38, ((11)*60+59)*60+31, ((11)*60+58)*60+24, ((11)*60+57)*60+17,
 ((12)*60+03)*60+59, ((12)*60+02)*60+49, ((12)*60+01)*60+54, ((12)*60+01)*60+52, ((12)*60+01)*60+32, ((12)*60+01)*60+22, ((12)*60+01)*60+12)).reshape(5,7)
print "Uhrzeiten in sec:\n",Uhrzeit_in_sec
# Jetzt müssen wir nur noch die Tag-zu-Tag Differenzen berechnen.
# Wenn wir ein '-' davor schreiben, dann erhalten wir ein positives Ergebnis, wenn
# am Tag darauf die Uhr nachgeht und ein negatives, wenn sie dann vorgeht.
ZeitDifferenzen = -diff(Uhrzeit_in_sec,1)
print "\nZeitdifferenzen:\n",ZeitDifferenzen
print "\nMittelwerte:\n",mean(ZeitDifferenzen,1)
print "\nStandardabweichungen:\n",std(ZeitDifferenzen,1)
# Dasselbe Ergebnis erhalten wir auch mit folgendem Befehl:
# print "\nStandardabweichungen:\n",sqrt(var(ZeitDifferenzen,1))
```

```
Uhrzeiten in sec:
[[45400 45416 45432 45447 45464 45479 45494]
 [43199 43202 43197 43207 43202 43196 43203]
 [57045 57045 57161 57219 57277 57335 57393]
 [43439 43372 43305 43238 43171 43104 43037]
 [43439 43369 43314 43312 43292 43282 43272]]

Zeitdifferenzen:
[[ -16  -16  -15  -17  -15  -15]
 [  -3    5  -10    5    6   -7]
 [   0 -116  -58  -58  -58  -58]
 [  67   67   67   67   67   67]
 [  70   55    2   20   10   10]]

Mittelwerte:
[-15.66666667  -0.66666667 -58.          67.          27.83333333]

Standardabweichungen:
[  0.74535599   6.3420992   33.48631561   0.          25.4323722 ]
```

Die 4. Uhr, d.h. Uhr 'D' geht zwar etwas nach (67 Sekunden), aber dafür in einer sehr konstanten Weise (Standardabweichung = 0).

Wenn wir dieses Nachgehen bereits wissen, dann können wir das sehr einfach korrigieren. Diese Uhr ist daher die präziseste.

Am unverlässlichsten ist die 3. Uhr ('C'), denn die Standardabweichung ihrer Zeitmessungen ist die größte.

Wir können also ein eindeutiges Ranking festlegen:

Platz 1: D

Platz 2: A

Platz 3: B

Platz 4: E

Platz 5: C