

1 Was ist Programmieren?

Programmieren ist ... wenn du einem Computer Anweisungen gibst und er tut, was du willst. Das klingt einfacher, als es ist. Aber keine Angst, Computern die richtigen Anweisungen zu geben, kann jeder lernen. Zunächst musst du dazu eine Sprache »sprechen«, die dein Computer auch versteht. Du hast dich dafür entschieden, den langen Weg zur Programmiererin oder zum Programmierer mit der Programmiersprache Python zu beginnen. Das wird dir deinen Weg sehr erleichtern! Das erste Stück dieses Weges werden wir gemeinsam gehen.

In diesem Kapitel wirst du ...

- erfahren, mit welcher Software du arbeitest, um mit Python zu programmieren.
- mit dieser Software deine ersten Erfahrungen sammeln.
- den interaktiven Python-Interpreter kennen lernen und beginnen, mit ihm die Programmiersprache Python zu erforschen.
- mit Python rechnen und schreiben.
- schließlich dein erstes Programm schreiben und gleich noch weiterentwickeln.

Wozu dienen Programmiersprachen?

Programmiersprachen sind dazu gemacht, einem Computer Anweisungen zu geben, damit er bestimmte Aufgaben ausführen kann. Solche Aufgaben sind etwa: Eine Meldung anzeigen, eine Rechnung ausführen, die dir selbst zu langweilig ist, oder eine bunte Grafik zeichnen.

Leistungsfähige Computerprogramme – Textverarbeitungsprogramme, Bildbearbeitungsprogramme, Computerspiele – bestehen aus einer Vielzahl solcher Anweisungen, sogar bis zu mehreren Millionen. Gemeinsam ist allen Programmen, kleinen wie großen, dass die Anweisungen nach ganz genauen Regeln aufgeschrieben werden müssen.

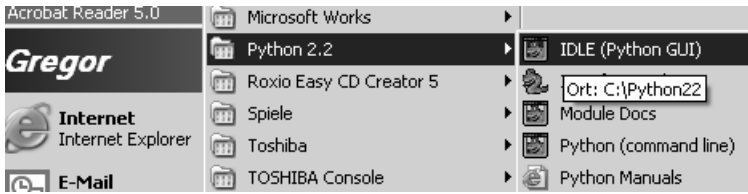
In diesem ersten Kapitel wirst du einige Anweisungen der Programmiersprache Python kennen lernen und ausprobieren – ja, gleich ausprobieren, denn das geht mit Python ganz leicht. Am Ende wirst du dann aus ein paar solchen Anweisungen dein erstes Programm zusammenbasteln.

Nun ist es aber so, dass die CPU deines Computers, sein eigentliches Herzstück und Arbeitstier, nicht Python kann. Die CPU ist ein Mikroprozessor, also eine Maschine und versteht nur *Maschinensprache*, z.B: 00001010011010110101001010100111. *Maschinensprache* braucht nur zwei Zeichen, 0 und 1. Das ist ihr ganzes Alphabet. Und weil es nur aus zwei Zeichen besteht, ist dieses Alphabet so einfach, dass die vielen Millionen elektronischen Schalter in der CPU in rasender Geschwindigkeit Anweisungen abarbeiten können: Strom ein – Strom aus.

Damit dein Computer aber deine Python-Anweisungen verstehen und ausführen kann, braucht er einen Helfer, der ihm diese Anweisungen in *Maschinensprache* übersetzt. Dieser Helfer ist selbst ein Computerprogramm und bei der Software dabei, die du mit dem Python-System auf deinem Computer installiert hast: Der Name des Programms ist – wenig verwunderlich – Python und es wird im Computer-Fachchinesisch als *Python-Interpreter* bezeichnet.

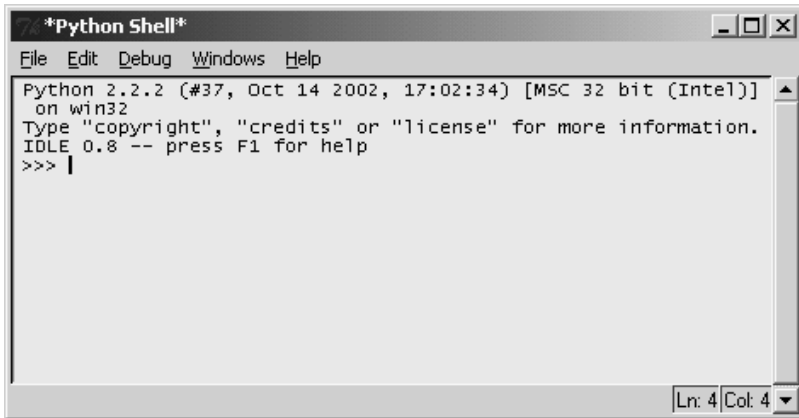
Unser Werkzeug: die IDLE

Nach der Installation von Python (siehe Anhang A) findest du in deinem Startmenü START|PROGRAMME|PYTHON2.2|IDLE (PYTHON GUI).



Die IDLE wird vom Startmenü aus gestartet.

→ Starte IDLE (Python GUI). Nun erscheint folgendes Fenster auf dem Bildschirm:



Die integrierte Entwicklungsumgebung IDLE

IDLE ist ein Werkzeug zum Programmieren mit Python. IDLE ist die Abkürzung für »**I**ntegrated **D**evelopment **E**nvironment«, auf Deutsch: *integrierte Entwicklungsumgebung*. Hat aber daneben noch andere Bedeutungen. (Sieh mal in einem Englischwörterbuch nach oder schau dir eine Folge von Monty Python's Flying Circus an!)

»Integriert« bedeutet, dass mehrere Bestandteile zu einem Werkzeug zusammengesetzt sind. Während unserer Arbeit mit Python werden wir häufig mit IDLE arbeiten. Das Schönste an IDLE ist, dass sie sich gleich mit einem interaktiven Python Interpreter meldet, der

>>>

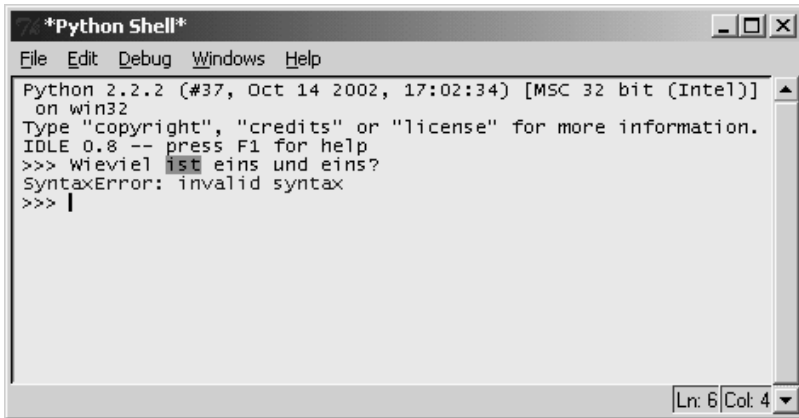
ins *PYTHON SHELL*-Fenster schreibt. Das ist sein Bereitschaftszeichen. Heute sagt man dazu meistens *Prompt*, weil sich dieses Wort, das im Englischen für das Bereitschaftszeichen verwendet wird, bereits im Deutschen eingebürgert hat. Rechts vom Prompt siehst du einen blinkenden Cursor |. So zeigt der interaktive Python-Interpreter an, dass er darauf wartet, dass du etwas eingibst.

Sehen wir mal nach, ob er wenigstens ganz einfache Dinge versteht.

→ Tippe folgende Frage ein:

```
>>> Wie viel ist eins und eins?
```

Oh weh!

A screenshot of a Python Shell window titled '*Python Shell*'. The window has a menu bar with 'File', 'Edit', 'Debug', 'Windows', and 'Help'. The main text area shows the following text:

```
Python 2.2.2 (#37, Oct 14 2002, 17:02:34) [MSC 32 bit (Intel)]  
on win32  
Type "copyright", "credits" or "license" for more information.  
IDLE 0.8 -- press F1 for help  
>>> Wieviel ist eins und eins?  
SyntaxError: invalid syntax  
>>> |
```

The status bar at the bottom right shows 'Ln: 6 | Col: 4'.

Syntax ist die Rechtschreibung bei einer Programmiersprache.

Wir haben im Eifer des Gefechts übersehen, dass der interaktive Python-Interpreter nicht Deutsch kann, sondern nur Python. Die gestellte Frage hält nun mal nicht die »Rechtschreibregeln« von Python ein. Im Computer-Kauderwelsch sagt man: Die Eingabe entspricht nicht der *Syntax* von Python. Deswegen beklagt der interaktive Python-Interpreter einen *Syntaxfehler!*

→ Also fragen wir anders herum:

```
>>> 1 + 1  
2  
>>>
```

Hmmm! Das hat der interaktive Python-Interpreter verstanden.

1+1 ist offenbar ein gültiger Python-*Ausdruck*. Muss wohl daran liegen, dass die Sprache der Mathematik international ist.

Kann der interaktive Python-Interpreter auf diesem Sektor noch mehr?

Die Arbeit mit dem IPI

Da fällt mir auf: »interaktiver Python-Interpreter« ist wirklich keine praktische Bezeichnung. Weil wir aber diesen interaktiven Python-Interpreter so oft verwenden werden, gebe ich ihm hier einen Spitznamen: IPI.

Der IPI meldet sich also im `*PYTHON SHELL*`-Fenster mit dem Bereitschaftszeichen `>>>`. Die Arbeit mit dem IPI besteht darin, dass man nach dem Bereitschaftszeichen einen *Python-Ausdruck* oder eine *Python-Anweisung* eingibt. Der IPI wertet die Eingabe aus und schreibt dann darunter eine Antwort in das `*PYTHON SHELL*`-Fenster. Auf diese Weise kannst du lernen und erforschen, wie Python funktioniert.



Du wirst im Folgenden viele »interaktive Übungen« mit dem IPI machen. Ich schreibe dann einfach:

→ Mach mit!

Dann kommt immer eine Folge von Eingaben für den IPI, die hinter dem Prompt

`>>>`

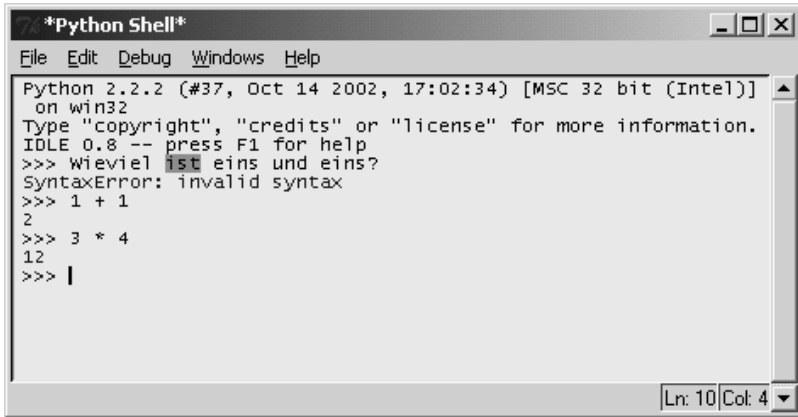
beginnen. Dazwischen streue ich Bemerkungen und Erklärungen ein. Ich meine, das Beste ist, wenn du diese Eingaben einfach eintippst und prüfst, ob sie bei dir dieselben Ergebnisse liefern. Auf diese Weise lernst du, wie man Python-Anweisungen richtig schreibt und wahrscheinlich auch, welche Fehler man leicht macht und wie man sie vermeidet. Wenn du dann auch noch darüber nachdenkst, warum deine Eingaben gerade die Ergebnisse liefern, die im Fenster erscheinen, dann lernst du auf diese Weise Python verstehen.

Also machen wir einfach unsere erste Sitzung mit dem IPI!

Rechnen

→ Mach mit!

```
>>> 3 * 4
12
```



```
>>> 13 + 4 * 3
25
>>> (13 + 4) * 3
51
>>> (3 - 5) * (13 + 4)
-34
```

Mit *arithmetischen Ausdrücken* kann der IPI offenbar gut umgehen. Kennt sogar die Vorrangregeln und die Klammernregel. Ist mindestens genau so gut wie dein Taschenrechner!

Der IPI kennt folgende Rechenzeichen:

+ - * / // % **

Du wirst sie alle in diesem Buch noch kennen lernen. Wenn du Lust hast, kannst du aber jetzt schon mit ihnen experimentieren. Gib dem IPI einfach ein paar Ausdrücke mit diesen Rechenzeichen ein. Vielleicht findest du heraus, zu welchen Rechenoperationen sie gehören?

Ob der IPI auch Wurzeln ziehen kann?

Na ja – so viel Mathe braucht man ja nicht immer. Aber doch immer wieder. Deshalb hält Python einige mathematische Funktionen nicht ständig bereit, sondern hat sie in einem eigenen *Modul* zusammengefasst, das bei Bedarf geladen werden kann. Dieses Modul heißt *math*. Dort ist auch die Quadratwurzel drin. Wenn du sie verwenden willst, musst du die Funktionen aus die-

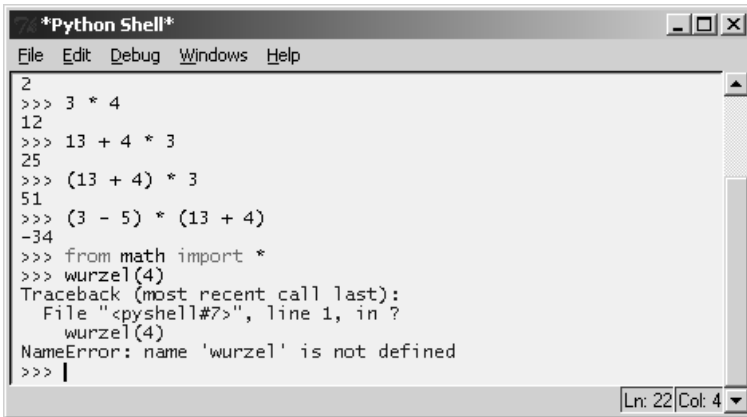
sem Modul *importieren*, das heißt: in den Arbeitsspeicher laden. Das geschieht mit der folgenden Python-Anweisung:

```
>>> from math import *
```

Beachte die Kleinschreibung (genauere Erklärung weiter hinten in diesem Kapitel)!

Nun berechnen wir die Wurzel aus 4:

```
>>> wurzel(4)
```



```
*Python Shell*
File Edit Debug Windows Help
2
>>> 3 * 4
12
>>> 13 + 4 * 3
25
>>> (13 + 4) * 3
51
>>> (3 - 5) * (13 + 4)
-34
>>> from math import *
>>> wurzel(4)
Traceback (most recent call last):
  File "<pysshell#7>", line 1, in ?
    wurzel(4)
NameError: name 'wurzel' is not defined
>>> |
```

Eine Fehlermeldung: Die Art des Fehlers steht in der letzten Zeile.

Oh, nein! Schon wieder eine *Fehlermeldung*! Damit musst du umgehen lernen! Fehlermachen ist beim Programmieren so unvermeidlich und gleichzeitig so wichtig fürs Weiterkommen wie beim Skaten!

Für den Anfang wird es wohl das Beste sein, dass du dir in solchen Fehlermeldungen nur die letzte Zeile ansiehst:

```
NameError: name 'wurzel' is not defined
```

Das erste Wort gibt immer die Art des Fehlers an: `NameError`. Das heißt, dass in der Eingabe ein *Name* vorgekommen ist, den der IPI nicht kennt. Und dann schreibt er uns auch noch hin, welchen Namen er nicht kennt: `wurzel!`



Also ist alles meine Schuld! Ich hätte dir gleich sagen sollen, dass sich die Bezeichnungen in Python – wie in fast allen Programmiersprachen – aus dem Englischen ableiten. Und im Englischen heißt *Quadratwurzel* `squareroot`. Die entsprechende Python-Funktion heißt aber `sqrt`, damit du weniger tippen musst!

Also – nachdem alle Funktionen aus `math` nun schon importiert sind:

```
>>> sqrt(4)
2.0
>>>
```

Richtig! Und doch ist wieder was Neues dabei! Eine Kommazahl kommt als Ergebnis heraus! Also kann Python auch mit Kommazahlen rechnen?

Beachte: Das Komma ist ein Punkt, kein Beistrich!

```
>>> 3.125 * 0.8
2.5
>>> sqrt(2)
1.4142135623730951
>>> 2 * sqrt(2)
2.8284271247461903
>>>
```

Ist es egal, ob man ganze Zahlen oder Kommazahlen verwendet? Darüber werden wir uns später noch unterhalten müssen, doch sieh dir mal Folgendes an:

```
>>> sqrt(2)*sqrt(2)
2.0000000000000004
>>>
```

Dieses Beispiel zeigt dir, dass das Rechnen mit Kommazahlen auf Computern unvermeidlich mit kleinen Ungenauigkeiten verbunden ist. Sie entstehen aus Fehlern, die beim Auf- und Abrunden im Mikroprozessor auftreten. (Das gilt in gleichem Maße für deinen Taschenrechner, auch wenn der im obigen Fall den Rundungsfehler verschweigen würde, weil er ja nur 10 oder 12 Stellen anzeigt, aber intern mit mehr Stellen arbeitet.)

Da hab ich eine Idee: Hast du nicht zufällig noch eine unfertige Mathe-Hausaufgabe rumliegen? Ich kenne das, oft hat der Computer eine größere Anziehungskraft als Hausaufgaben. Wenn ja, dann hole sie dir rasch her und verwende zur Lösung der Aufgaben doch den IPI als Ersatz für deinen Taschenrechner – sehr praktisch, zwei Fliegen auf einen Schlag! Hausaufgabe erledigt und Python besser kennen gelernt.

Schreiben

```
>>> print "Hallo Große! Du wirst sehen, Python macht Spaß!"
Hallo Große! Du wirst sehen, Python macht Spaß!
>>>
```

Da haben wir schon wieder eine neue Python-Anweisung verwendet: die `print`-Anweisung. Sie dient zur Ausgabe von Ausdrücken auf dem Bildschirm. Hier hat sie einen Text ausgegeben. Texte sind Folgen von Zeichen (Buchstaben, Ziffern, Leer-, Satz- und Sonderzeichen), die zwischen Anführungszeichen stehen müssen. Dadurch weiß der IPI, dass diese Zeichen keine Namen oder Ausdrücke mit anderer Bedeutung sind, und schreibt sie einfach buchstäblich – also Zeichen für Zeichen – hin. Solche Folgen von Zeichen nennt man im Computerlatein *Zeichenketten* oder *Strings*. (Merkwürdig, dass das Computerlatein meistens aus dem Englischen kommt!)

```
>>> print "3*12"  
3*12  
>>> print 3*12  
36  
>>> print 36*1  
36  
>>> print 36  
36
```

Das ist schon ein wichtiger Unterschied: `"3*12"` ist ein String und wird von `print` Zeichen für Zeichen hingeschrieben. `3*12` ist dagegen ein arithmetischer Ausdruck. Er wird (vom IPI!) zuerst ausgerechnet und das Ergebnis wird dann auf dem Bildschirm ausgegeben. Natürlich sind `36*1` und auch nur `36` arithmetische Ausdrücke.

Zuletzt zeige ich dir noch, dass man mit `print` auch mehrere Dinge nebeneinander ausgeben kann. Diese Dinge müssen dann durch Beistriche getrennt nebeneinander geschrieben werden:

```
>>> print 1, 2, 3*4  
1 2 12  
>>> print "3*12 =", 36  
3*12 = 36  
>>> print "3*12 =", 3*12  
3*12 = 36  
>>> print "3 * 12 =", 3*12, "und 4 * 12 =", 4*12  
3 * 12 = 36 und 4 * 12 = 48
```

Zähl nach: Die letzte `print`-Anweisung hat vier Dinge auszugeben: zwei Strings und zwei Zahlen, die aus arithmetischen Ausdrücken berechnet werden.



Clara Pythias Python-Special


Vielleicht willst du einmal, dass `print` so etwas ausgibt:

Uwe rief "Oh!" und erbleichte.

So geht das leider nicht:

```
>>> print "Uwe rief "Oh!" und erbleichte."  
SyntaxError: invalid syntax  
>>>
```

Dieser Fehler kündigt sich bereits während der Eingabe durch die Farben an. Der IPI glaubt, dass schon vor `Oh!` der erste String endet, denn dort sind ja die schließenden Anführungszeichen. Und er weiß dann mit `Oh!` nichts mehr anzufangen.

Mit Python gibt's da einen leichten Ausweg: Du darfst Strings auch mit `'` begrenzen. Das einfache Hochkomma `'` ist das Zeichen, das auf den meisten Tastaturen auf einer Taste gemeinsam mit dem  zu finden ist.

```
>>> print 'Uwe!'  
Uwe!
```

Innerhalb solcher Strings können ohne Probleme `"` verwendet werden. Also:

```
>>> print 'Uwe rief "Oh!" und erbleichte.'  
Uwe rief "Oh!" und erbleichte.
```

Ich rate dir aber, wenn möglich, einheitlich die `"` zu verwenden.

Dein erstes Programm

Wir haben bisher ausschließlich im Direktmodus gearbeitet, das heißt: Wir haben unsere Anweisungen direkt in den IPI eingegeben. Diese Anweisungen wurden sofort ausgeführt. Wenn du heute mit vielen Anweisungen eine schöne Figur zeichnest, dann kannst du sie morgen niemandem mehr zeigen. Dafür gibt es die Möglichkeit, diese Anweisungen zu speichern. Das nennt man ein *Programm*.

Ich finde, dass es jetzt an der Zeit ist, dass du dein erstes Programm schreibst – auch wenn es nur einige `print`-Anweisungen enthalten kann, das ist ja immerhin schon etwas!

Hier geht's darum zu lernen, was ein Programm ist, wie man es schreibt und wie man es ausführt.

Um gleich von Anfang an etwas Ordnung in deine Programme zu bringen:

Lege ein Verzeichnis an, in das du deine Programme speichern willst. Vorschlag: Du erzeugst auf dem Laufwerk C: ein Verzeichnis mit dem Namen Py4Kids. Dort legst du dann nach Bedarf Unterverzeichnisse kap01, kap02 usw. an, in die deine Programme, die du zu den entsprechenden Kapiteln des Buches schreibst, hinein kommen.



Aufgabenstellung: Ich schlage vor, dass du ein Programm schreibst, das folgende Ausgabe erzeugt:

```
Hi Kleiner!  
Wie viel ist eins und eins?  
Ganz leicht!  
1 + 1 = 2
```

Wie könnte man so etwas angehen?

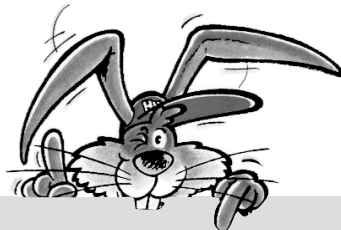
Einfache Python-Programme bestehen aus einer Folge von Python-Anweisungen, die als Programmtext in einer Datei gespeichert werden. Solche Programme werden auch oft als *Scripts* bezeichnet.

Mit der Entwicklungsumgebung IDLE erstellst du Python-Scripts mit folgenden Schritten.

→ Mach mit:

→ **Schritt 1:** Öffne ein »Editor-Fenster«. Dazu wähle das Menü FILE|NEW WINDOW oder drücke die Tastenkombination `Strg` + `N`.

Ein Editor-Fenster dient der Eingabe und Bearbeitung von Programmtexten.



Beachte! Ab sofort verwenden wir zwei Fenster: den IPI für den Direktmodus und das Editor-Fenster für die Programme, die wir schreiben.

Diese beiden verhalten sich ganz unterschiedlich. Der Hauptunterschied ist:

- Der IPI versteht Python und wertet deine Ausdrücke, einen nach dem anderen, aus. Die IDLE hat immer nur *ein* Fenster mit einem interaktiven Python-Interpreter.
- Ein Editor-Fenster dient nur zum Schreiben der Programme – es ist eigentlich nur ein kleines Textverarbeitungsprogramm. (Ein bisschen Python versteht auch ein Editor-Fenster. Das wirst du später sehen, wenn es dir hilft, Python-Scripts in der richtigen Form zu schreiben.) Du kannst in der IDLE gleichzeitig mehrere Editor-Fenster geöffnet haben.

→ **Schritt 2:** Schreibe ins Editor-Fenster die Programmanweisungen. In unserem Beispiel sind das folgende:

```
print "Hi Kleiner!"  
print "Wie viel ist eins und eins?"  
print "Ganz leicht!"  
print "1 + 1 =", 1 + 1
```



Achte darauf, dass der Text jeder Zeile *ganz links* beginnt! Leerzeichen vor einer einfachen Python-Anweisung sind hier nicht erlaubt.

Bereits in Kapitel 3 wirst du erfahren: In Python haben Leerzeichen am Anfang von Programmzeilen eine besondere *Bedeutung*. Es wird daher durch die Syntaxregeln von Python festgelegt, wo und zu welchem Zweck Leerzeichen hingehören.

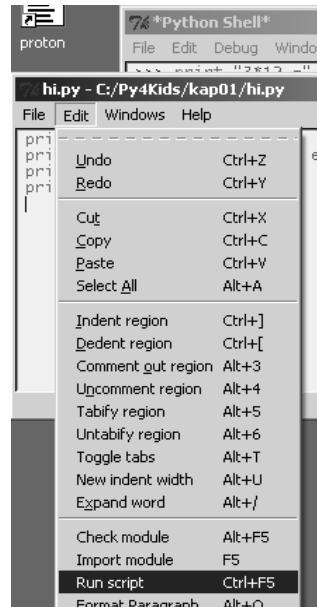
Falsche Leerzeichen führen zu Syntaxfehlermeldungen.

→ **Schritt 3:** Speichere das Programm unter einem geeigneten Namen, z.B. `hi.py` ab. Python-Programme müssen die Endung `.py` haben. Wähle dazu im Editor-Fenster das Menü FILE|SAVE AS... Noch schneller geht es mit der Tastenkombination `[Strg] + [S]`.

Dateinamen sollten einen Bezug zum Inhalt des Programms haben, damit du auch später noch leicht erkennen kannst, was das Programm macht.

→ **Schritt 4:** Führe das Programm in der IDLE aus: Wähle dazu *im Editor-Fenster* das Menü EDIT|RUN SCRIPT oder die Tastenkombination `[Strg] + [F5]`. (Dieses Kommando ist im Menü des *PYTHON SHELL*-Fensters nicht zu finden.)

- ➔ Solltest du vergessen haben, das Programm vor der Ausführung zu speichern, wirst du nun aufgefordert, das zu tun. Von der IDLE aus kann nur ein nach der letzten Änderung gespeichertes Programm ausgeführt werden.



Das Programm wird vom Editor-Fenster aus gestartet.

Ein Blick auf die Titel-Leiste des Editor-Fensters zeigt dir sofort, ob das Programm nach dem letzten Speichern geändert wurde: Dann findest du Sternchen vor und nach dem Titel. Diese Sternchen verschwinden beim Abspeichern.

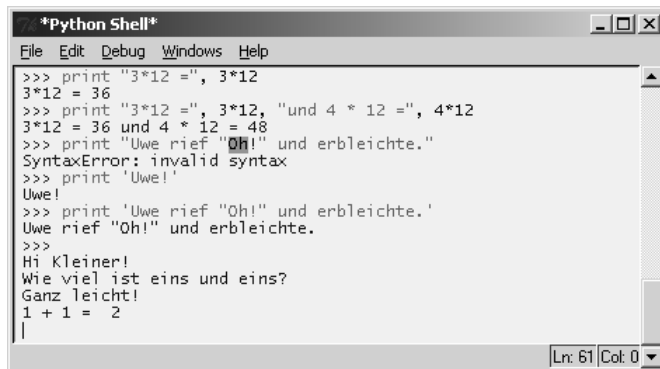


Programm geändert



Programm gespeichert

- ➔ Die IDLE benutzt nun ihren Python-Interpreter, um das eingegebene Programm auszuführen. Nun erscheint Folgendes im *PYTHON SHELL*-Fenster:



Das erste Programm wird ausgeführt.

Das ist die Ausgabe unseres Programms. Der Cursor blinkt unterhalb der letzten Ausgabezeile.

- **Schritt 5:** Drücke auf die Eingabetaste. Nun erscheint wieder die Bereitschaftsanzeige des IPI. Der Python-Interpreter, der eben noch dein Programm ausgeführt hat, kann nun wieder interaktiv verwendet werden.



Bevor wir die Arbeit mit diesem Programm abschließen, wollen wir noch in das Programm hineinschreiben, wer wann wozu dieses Programm gemacht hat. Natürlich wird das kein Text für den Python-Interpreter sein – sondern Text für dich, wenn du später mal das Programm wieder anschaust. Diese Kommentare dienen auch für deine Freunde, Lehrer usw. als Information.

In Python werden Kommentare durch das #-Zeichen markiert. Alles, was in einer Zeile hinter diesem Zeichen steht, wird vom Interpreter als Kommentar erkannt und ignoriert.

- **Schritt 6:** Schreibe an den Anfang deines Programms einen *Kopfkommentar*, bestehend aus drei Kommentarzeilen nach dem unten stehenden Muster:

```
# Autor: Gregor Lingl
# Datum: 27. 7. 2002
# hi.py: Mein erstes Programm

print "Hi, Kleiner!"
print "Wie viel ist eins und eins?"
print "Ganz leicht!"
print "1 + 1 =", 1 + 1
```

Listing 1: Programm hi.py

- **Schritt 7:** Speichere das Programm noch einmal ab! Jetzt reicht FILESAVE, weil das Programm schon einmal gespeichert wurde. Natürlich kannst du es jetzt noch einmal ausführen, aber an der Ausgabe ändert sich durch das Einfügen von Kommentaren nichts!

Wir erweitern unser erstes Programm

Oft wirst du bei der Arbeit mit diesem Buch vor der Aufgabe stehen, aus einem Programm, das du geschrieben hast, ein neues zu entwickeln. Im Folgenden möchte ich dir zeigen, wie man dabei vorgeht:

Aufgabenstellung: Wir wollen unser Programm `hi.py` so erweitern, dass es folgende Ausgabe erzeugt:

```
Hi!
Wie viel ist eins und eins?
Ganz leicht!
1 + 1 = 2

Und wie viel ist die Wurzel aus 4?
Nicht mehr ganz so leicht!
Wurzel aus 4 ist 2.0
```

Was man dazu braucht, haben wir weiter oben schon alles ausprobiert. Ausgenommen: Wie erzeugt man eine Leerzeile? Wir werfen den IPI an und probieren:

→ Mach mit!

```
>>> print "irgendwas"
irgendwas
>>> print ""

>>> print

>>>
```

Da haben wir gleich zwei Möglichkeiten zur Auswahl: Entweder wir lassen `print` einen so genannten *Leerstring* ausgeben, also zwei Anführungszeichen ohne etwas dazwischen: `""`. Das ist sozusagen eine Zeichenkette ohne Zeichen. (Erinnert irgendwie an die leere Menge aus Mathe ...) Oder wir schreiben überhaupt nur `print`.

Apropos Mathe! Um die Quadratwurzel auszurechnen, brauchen wir die Funktionen aus `math`. Daher muss unser Programm diese auch importieren! Wie die `import`-Anweisung aussieht, wissen wir ja schon von unserer IPI-Sitzung. Man kann sie genauso in ein Programm übernehmen. Üblicherweise schreibt man sie an den Anfang eines Programms.

Beginnen wir. Ich gehe davon aus, dass die IDLE gestartet ist, dass aber kein Editor-Fenster geöffnet ist.

- ➔ Öffne vom *PYTHON SHELL*-Fenster aus dein erstes Programm `hi.py`. (Menü FILE|OPEN...)
- ➔ Wir wollen dem neuen Programm den Namen `himath.py` geben, um es daran zu erinnern, dass es das Modul `math` benutzt. Ein Schelm, wer da gleich an höhere Mathematik denkt! Also ändere im Kopfkomentar den Programmnamen dementsprechend und schreibe daneben: Mein zweites Programm. Dann speicherst du das Ganze unter dem neuen Namen ab.
- ➔ Unter den Kopfkomentar schreibe als erste Anweisung:

```
from math import *
```

Nach den bereits vorhandenen vier `print`-Anweisungen sind jetzt noch weitere vier `print`-Anweisungen anzufügen. Die erste soll eine Leerzeile erzeugen, die nächsten beiden sollen bestimmte Strings ausgeben. Welche das sind, kannst du der Aufgabenstellung zu dieser Übung sofort entnehmen. Die letzte `print`-Anweisung muss einen String und das Ergebnis der Wurzelberechnung ausgeben.

- ➔ **Schritt 1:** Füge die vier `print`-Anweisungen an den Programmtext an.
- ➔ **Schritt 2:** Sichere das erweiterte Programm (hier bewährt sich die Tastenkombination `[Strg] + [S]` im *Editor-Fenster*).
- ➔ **Schritt 3:** Führe das Programm mit `[Strg] + [F5]` ebenfalls im *Editor-Fenster* aus.

Hat alles geklappt? Sollte sich ein Fehler eingeschlichen haben, wird im *PYTHON SHELL*-Fenster eine Fehlermeldung erscheinen. Ich führe das hier mal an einem Beispiel vor. Angenommen, du hättest geschrieben:

```
print "1 + 1 =", 1 + 1
Print
print "Und wie viel ist die Wurzel aus 4?"
```

Dann hätte der Versuch, das Programm auszuführen, Folgendes ergeben:

```
>>>
Hi!
Wie viel ist eins und eins?
Ganz leicht!
1 + 1 = 2
Traceback (most recent call last):
  File "C:/Py4Kids/kap01/himath.py", line 11, in ?
    Print
NameError: name 'Print' is not defined
```

Aha! Auf einmal ist `print` »nicht definiert!« Erst auf den zweiten Blick bemerkst du, dass in der angeführten Zeile 11 das Wort `Print` großgeschrieben steht. Und das ist für Python ein anderes Wort als das kleingeschriebene `print`.

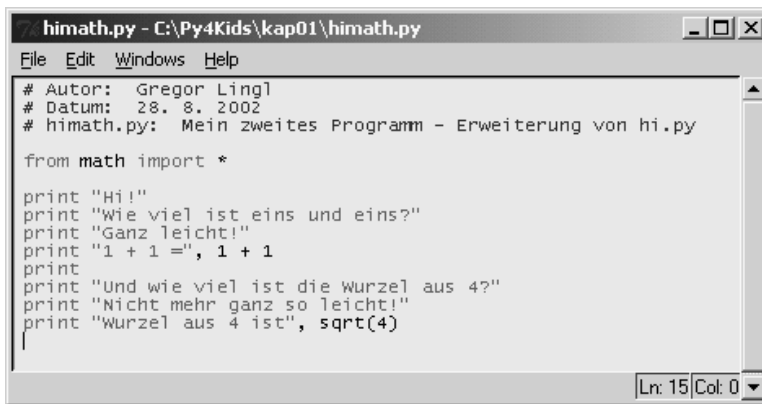
Python unterscheidet Groß- und Kleinschreibung! Das hat es mit anderen wichtigen Programmiersprachen wie C, C++ und Java gemeinsam!

Wörter, die sich nur in der Groß-/Kleinschreibung unterscheiden, sind für Python *verschiedene* Wörter!



Die nächsten Schritte: Fehler ausbessern und nochmals laufen lassen, bis keine Fehler mehr auftreten.

Weil du ja noch ganz am Anfang stehst, gebe ich dir zum Vergleich hier noch meine Lösung dieser Übung an. Ähnelt sie deiner?



```
himath.py - C:\Py4Kids\kap01\himath.py
File Edit Windows Help
# Autor: Gregor Lingl
# Datum: 28. 8. 2002
# himath.py: Mein zweites Programm - Erweiterung von hi.py

from math import *

print "Hi!"
print "Wie viel ist eins und eins?"
print "Ganz leicht!"
print "1 + 1 =", 1 + 1
print
print "Und wie viel ist die Wurzel aus 4?"
print "Nicht mehr ganz so leicht!"
print "Wurzel aus 4 ist", sqrt(4)
|
Ln: 15 | Col: 0
```

Wurde das Programm `himath.py` im Editor-Fenster nach der letzten Änderung gespeichert?

Syntax-Colouring: bunte Farben für die Syntax

Bei der Arbeit mit der IDLE wird dir aufgefallen sein, dass der Text in unterschiedlichen Farben erscheint. Wörter wie `print` oder `import` sind orange, Strings sind grün.

Das liegt daran, dass die IDLE die Python-Syntax kennt. Sie weiß, dass Strings in Python-Programmen eine besondere Rolle spielen, sie kennt die so genannten *reservierten Wörter* von Python und einiges mehr.

Auf diese Weise kann die IDLE die Struktur deiner Programme mit Farben verdeutlichen.

Diese Einfärbung der Wörter gemäß der Syntax von Python nennt man *Syntax-Colouring*. Leider können die Bilder in diesem Buch dies nur unvollkommen durch unterschiedliches Grau darstellen.

Gewöhne dir an, darauf zu achten. Denn dann kann dir so ein Fehler wie Großschreibung eines reservierten Wortes nicht passieren. Ein `Print` färbt die IDLE nicht orange. Da muss also was faul sein.

Reservierte Wörter werden für grundlegende Bestandteile der Sprache Python verwendet. Dazu gehören die `print`-Anweisung und die `import`-Anweisung. Beachte, dass auch das Wort `from` ein reserviertes Wort ist.



Es gibt in Python nur insgesamt 29 reservierte Wörter. Sie dürfen für keinerlei andere Zwecke verwendet werden als für die, für die sie in Python vorgesehen sind.

Reservierte Wörter, die wir bereits kennen: `print`, `from`, `import`.

Mit Mustern arbeiten

Programmierer arbeiten sehr viel mit *Mustern*. Manche Muster betreffen ganze Programme oder große Teile von Programmen, andere Muster betreffen kleine Bestandteile.

Das funktioniert im Wesentlichen so: Du stehst vor einer Aufgabe und suchst in deinem Kopf nach einem dir bekannten Denk-, Programm- oder Anweisungsmuster, das für die Lösung der Aufgabe angewendet werden kann. Jedenfalls ist es immer leichter, wenn du eine Aufgabe mit einem vertrauten Muster lösen kannst, als wenn du einen neuen Lösungsweg finden musst.

Deshalb werde ich dir für immer wieder vorkommende Problemstellungen Muster formulieren. Versuche, sie in deinem Kopf abrufbar zu verankern.

Sicher wirst du bei der Arbeit mit diesem Buch noch einige einfache Python-Scripts schreiben, daher gebe ich dir dafür gleich ein einfaches Muster:

Muster 1: Einfaches Python-Script

```
# Autor: Wer
# Datum: Wann
# Dateiname: Was    [Kommentare]
from Modul import * [Wird nicht in jedem Programm gebraucht]

Anweisung 1
Anweisung 2
...
```

[Leere Zeilen können nach Belieben eingeschoben werden, um das Programm leichter lesbar zu machen.]

Die in dieser Musterbeschreibung *kursiv* geschriebenen Wörter werden durch konkrete Informationen ersetzt. Im obigen Muster z. B. *Wer*. Ich schreibe dafür Gregor Lingl hinein. Du eben deinen Namen.

Oder *Modul*: Da kommt es drauf an, welches Script du schreibst. Willst du etwas Kompliziertes rechnen, musst du dort *math* einsetzen. Willst du etwas zeichnen, dann brauchst du auch *dafür* ein passendes Modul. Manche Programme – wie unser *hi.py* – müssen gar kein Modul importieren.

Zusammenfassung

- Um Python-Programme auszuführen, braucht man einen Übersetzer in die Maschinensprache: den Python-Interpreter.
- Anweisungen müssen die Rechtschreibregeln der Sprache Python einhalten.
- Die Rechtschreibregeln einer Programmiersprache nennt man auch ihre Syntax.
- Um Python-Programme zu schreiben, verwendet man ein Entwicklungswerkzeug: die IDLE.
- Die IDLE enthält einen interaktiven Python-Interpreter. Er führt direkt und sofort einzelne Python-Anweisungen aus und schreibt Ergebnisse an.
- Die IDLE enthält auch einen Editor. Das ist ein Fenster zum Verfassen und Bearbeiten von Programmtexten.
- Python kann rechnen und verwendet dafür die uns bekannte mathematische Schreibweise.

- Für gehobeneren Rechnungen muss Python Funktionen aus dem Modul `math` importieren.
- Beim Rechnen mit Kommazahlen macht Python Rundungsfehler.
- Python kann auch Texte (Strings genannt) schreiben.
- Python-Programme sind Dateien, die eine Folge von Python-Anweisungen enthalten.
- Python-Programme können durch `#` gekennzeichnete Kommentare enthalten.
- Am Kopf jedes Programms sollte ein Kommentar mit Angaben zu Autor, Erstellungsdatum und Programmzweck stehen.
- Python-Programme können innerhalb der IDLE ausgeführt werden. Dazu wird im Editor-Fenster das Menü `EDIT|RUN SCRIPT` oder die Tastenkombination `Strg + F5` benutzt.
- Die IDLE kennt die Python-Syntax und kennzeichnet verschiedene Bestandteile eines Programms durch Farben.

Zum Abschluss noch zwei Übungsaufgaben ...

Vielleicht hast du es übertrieben gefunden, den Python-Interpreter ausrechnen zu lassen, wie viel $1 + 1$ ist. Vielleicht dachtest du gar, man hätte als letzte Anweisung in `hi.py` schreiben können:

```
print "1 + 1 = 2"
```

Wenn du dir das gedacht hast, dann hast du Recht damit. Die Ausgabe hätte dann genauso ausgesehen.

Und trotzdem ist es gut, dass du Python rechnen lässt, wenn Python schon rechnen kann:

Aufgabe 1: Ändere `himath.py` so zu `himath2.py` ab, dass es die Quadratwurzel von `152399025` ausgibt.

Aufgabe 2: Erweitere `himath2.py` oder auch `hi.py` so, dass das Programm nicht nach $1 + 1$, sondern nach $123456789 * 987654321$ fragt und dieses Produkt dann auch ausgibt. Speichere dieses Programm unter einem neuen Namen ab.

Wenn dir die Lösung von Aufgabe 2 gelungen ist, dann wirst du merken, dass Python im Rechnen höchstwahrscheinlich besser ist als dein Taschenrechner!

... und ein paar Fragen

- Was ist der Unterschied zwischen `print "1+2"` und `print 1+2`?
- Mit welcher Funktion berechnet man in Python Quadratwurzeln?
- Was ist der Unterschied zwischen dem `*PYTHON SHELL*`-Fenster und einem Editor-Fenster?
- Rätsel: Wie erzielst du folgende Bildschirmausgabe:

```
Sie sagte: "No, don't do it."
```

Zur Vertiefung ...

... findest du auf der Buch-CD in der Datei `vertiefung01.pdf` Folgendes:
Bei der Arbeit mit dem IPI ist dir sicher so etwas untergekommen:

```
>>> 2 * 1.4
2.7999999999999998
>>> print 2 * 1.4
2.8
```

Hier erfährst du etwas über die Ursachen für diesen merkwürdigen Unterschied.