

HANS-GEORG SCHUMANN

PYTHON

FÜR **KIDS**

3. AUFLAGE

PROGRAMMIEREN LERNEN
OHNE VORKENNTNISSE



mitp

INHALT

EINLEITUNG	11
Was heißt eigentlich Programmieren?	11
Was ist eine Entwicklungsumgebung?	12
Warum gerade Python?	12
Die Entwicklungsumgebung	13
Wie arbeite ich mit diesem Buch?	14
Was brauchst du für dieses Buch?	15
Hinweise für Lehrer	16

ERSTE SCHRITTE	17
Mit Python loslegen	18
Zahlen und Text	21
Eine Arbeitsumgebung namens IDLE	25
Die erste py-Datei	28
Quelltext-Spielereien	32
Python verlassen	34
Zusammenfassung	35
Ein paar Fragen	36
... aber noch keine Aufgabe	36

1

BEDINGUNG UND KONTROLLE	37
Die if-Struktur	38
if und else	42
Ein bisschen Grundrechnen	46
Was für Zahlen?	49
Die Sache mit try und except	52
Zusammenfassung	54
Ein paar Fragen	55
... und ein paar Aufgaben	55

2

3	VERGLEICHEN UND WIEDERHOLEN	57
	Zensurenbild	58
	Ein kleines Ratespiel	62
	Dein Computer zählt mit	67
	Noch mehr Spielkomfort	69
	Zusammenfassung	71
	Ein paar Fragen ...	72
	... und ein paar Aufgaben	72
4	GELD-SPIELEREIEN	73
	Spiel mit dem Glück	74
	Die for-Struktur	76
	Auf dem Weg zum Millionär	79
	Macht Lotto reich?	83
	Zeichen-Verkettung	86
	Zusammenfassung	88
	Ein paar Fragen ...	89
	... und ein paar Aufgaben	89
5	FUNKTIONEN	91
	Python ist lernfähig	92
	Lokal oder global?	95
	Parameter	97
	Tauschprozesse	100
	Zahlen sortieren	104
	Zusammenfassung	107
	Ein paar Fragen ...	107
	... und ein paar Aufgaben	107
6	KLASSEN UND MODULE	109
	Ein neues Baby?	110
	self und __init__	113
	Vererbung	116
	Programm-Module	120
	Privat oder öffentlich?	125
	Zusammenfassung	129
	Ein paar Fragen ...	129
	... aber keine Aufgabe	129

EINSTIEG IN TKINTER	131
Erst mal ein Fenster	132
Es passiert etwas	135
Layout-Management	138
Meldeboxen und Titelleisten	141
Alles noch mal: als Klasse	142
Zusammenfassung	143
Ein paar Fragen	144
... und eine Aufgabe	144

7

KOMPONENTEN-SAMMLUNG	145
Kleine Button-Parade	145
Antwort-Knöpfe und Diagnose-Felder	148
Listenwahl	151
Von Pünktchen	153
... und Häkchen	156
Verschönerung	159
Zusammenfassung	162
Ein paar Fragen	163
... und ein paar Aufgaben	163

8

AKTION SEELENKLEMPNER	165
Klempner-Bauplan	166
Bereit zur Diagnose?	169
Datentransfer	172
Alles zusammen	176
Therapieprotokoll	178
Zusammenfassung	180
Ein paar Fragen	180
... und ein paar Aufgaben	181

9

MENÜS UND DIALOGS	183
Ein Menü für den Klempner	184
Zwei Dialoge	186
Alles zusammen	189
Pop it up!	191
Shortcuts gefällig?	194

10

Zusammenfassung	196
Ein paar Fragen	197
... aber keine Aufgabe	197

11	GRAFIK IN PYTHON	199
	Von Punkten und Koordinaten	199
	Das erste Bild	202
	Jetzt wird's bunt	205
	Eckig und rund	207
	Mit Text geht auch	209
	Farbtupfer	210
	Selber zeichnen?	212
	Turtle-Grafik	214
	Zusammenfassung	217
	Ein paar Fragen	218
	... und ein paar Aufgaben	218

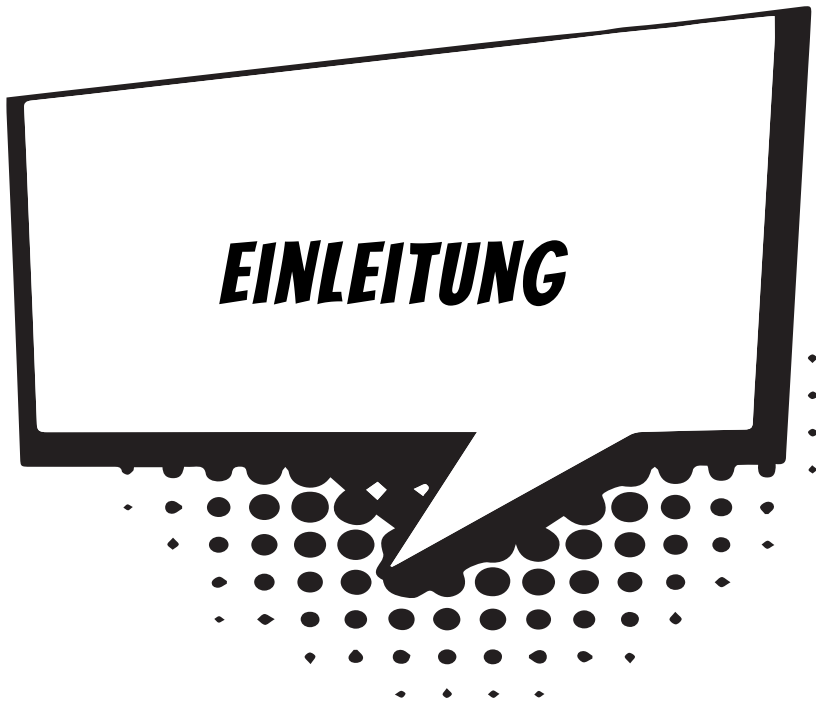
12	ANIMATIONEN	219
	Erst mal ein Kreis	219
	Canvas ruft Image	223
	Bildersammlung	225
	Eine Player-Klasse	228
	Wie läuft's?	231
	Drehungen	233
	Verschwinden und auftauchen	235
	Zusammenfassung	238
	Eine Frage	238
	... und ein paar Aufgaben	238

13	KLEINER KRABELKURS	239
	Einstieg mit Pygame	240
	Ein Objekt im Fenster	242
	Insekt als Player	245
	Tastensteuerung	250
	Drehmomente	253
	Grenzkontrollen	257
	Zusammenfassung	258

Ein paar Fragen ...	259
... und eine Aufgabe	260
 VOM KÄFER ZUR WANZE	 261
Die Sache mit der Maus	262
Ohne Mathe geht es nicht	263
Alles zusammen	268
Freilauf	269
Klick und Platt	273
Klassentrennung	275
Zusammenfassung	278
Ein paar Fragen ...	279
... und eine Aufgabe	279
 DODGE ODER HIT	 281
Ein neuer Player	282
Stand, Duck, Jump	285
Die Ding-Klasse	288
Ausweichmanöver	291
Das Hauptprogramm	295
Zusammenfassung	296
Keine Fragen ...	297
... und nur eine Aufgabe	297
 PLAY THE GAME	 299
Punkte sammeln	300
Eine Game-Klasse	303
Wanzen-Sammlung	306
Killer-Punkte	309
Zusammenfassung und Schluss	312
 ANHANG	 315
Python installieren	315
Pygame installieren 1	319
Pygame installieren 2	323
Einsatz der Buch-Dateien	324

14**15****16****A**

B	ANHANG	325
	Kleine Checkliste	325
	STICHWORTVERZEICHNIS	327



Python – wer denkt da nicht an eine Schlange? Eine, die zwar nicht giftig ist, aber einem sämtliche Knochen brechen und die Luft abschnüren kann. Du weißt natürlich, dass es hier bei Python um eine Programmiersprache geht.

Python wurde Anfang der 1990er-Jahre entwickelt. Der Name dieser Sprache geht jedoch nicht auf die gleichnamige Schlange zurück, sondern auf eine Truppe von englischen Komikern namens »Monthy Python«, die vor allem in den 70er-Jahren mit ihren skurrilen Filmen erfolgreich war (unter anderem mit »Das Leben des Brian«).

Natürlich hat Python viel von anderen Programmiersprachen übernommen. Sie hat sich einen Namen gemacht, weil sie als leicht erlernbar und übersichtlich gilt. In diesem Buch geht es um die bereits dritte Version von Python, die aktuell auch die neueste (und vielseitigste) ist.

WAS HEIßT EIGENTLICH PROGRAMMIEREN?

Wenn du aufschreibst, was ein Computer tun soll, nennt man das Programmieren. Das Tolle daran ist, dass du selbst bestimmen kannst, was getan werden soll. Lässt du dein Programm laufen, macht der Computer die Sachen, die du ausgeheckt hast.

Natürlich wird er dann dein Zimmer nicht aufräumen und dir auch keine Tasse Kakao ans Bett bringen. Aber beherrscht du erst mal das Programmieren, kannst du den Computer sozusagen nach deiner Pfeife tanzen lassen.

Allerdings passiert es gerade beim Programmieren, dass der Computer nicht so will, wie du es gerne hättest. Meistens ist das ein Fehler im Programm. Das Problem kann aber auch irgendwo anders im Computer oder im Betriebssystem liegen. Das Dumme bei Fehlern ist, dass sie sich gern so gut verstecken, dass die Suche danach schon manchen Programmierer zur Verzweiflung gebracht hat.

Vielleicht hast du nun trotzdem Lust bekommen, das Programmieren zu erlernen. Dann brauchst du ja nur noch eine passende Entwicklungsumgebung, und schon kann's losgehen.

WAS IST EINE ENTWICKLUNGSUMGEBUNG?

Um ein Programm zu erstellen, musst du erst einmal etwas eintippen. Das ist wie bei einem Brief oder einer Geschichte, die man schreibt. Das Textprogramm dafür kann sehr einfach sein, weil es ja nicht auf eine besondere Schrift oder Darstellung ankommt wie bei einem Brief oder einem Referat. So etwas wird **Editor** genannt.

Ist das Programm eingetippt, kann es der Computer nicht einfach lesen und ausführen. Jetzt muss es so übersetzt werden, dass der PC versteht, was du von ihm willst. Weil er aber eine ganz andere Sprache spricht als du, muss ein Dolmetscher her.

Du programmierst in einer Sprache, die du verstehst, und der Dolmetscher übersetzt es so, dass es dem Computer verständlich wird. So etwas heißt dann **Compiler** oder **Interpreter**.

Python bietet solche Dolmetscher gleich für mehrere Betriebssysteme. Dein Computer kann also ein Windows-PC oder ein Linux-PC sein, ein Macintosh oder irgendein anderes Computersystem. Ein und dasselbe Python-Programm kann so (eventuell mit kleinen Abweichungen) auf jedem beliebigen Computer funktionieren.

Schließlich müssen Programme getestet, überarbeitet, verbessert, wieder getestet und weiterentwickelt werden. Dazu gibt es noch einige zusätzliche Hilfen. Daraus wird dann ein ganzes System, die Entwicklungsumgebung.

WARUM GERADE PYTHON?

Leider kannst du nicht so programmieren, wie dir der Schnabel gewachsen ist. Eine Programmiersprache muss so aufgebaut sein, dass möglichst viele Menschen in möglichst vielen Ländern einheitlich damit umgehen können.

Weil in der ganzen Welt Leute zu finden sind, die wenigstens ein paar Brocken Englisch können, besteht auch fast jede Programmiersprache aus englischen Wörtern. Es gab auch immer mal Versuche, z.B. in Deutsch zu programmieren, aber meistens klingen die Wörter dort so künstlich, dass man lieber wieder aufs Englische zurückgreift.

Eigentlich ist es egal, welche Programmiersprache du benutzt. Am besten eine, die möglichst leicht zu erlernen ist. Wie du weißt, bekommst du es in diesem Buch mit der Programmiersprache Python zu tun, die mittlerweile sehr weit verbreitet ist. (Willst du mal in andere Sprachen hineinschnuppern, dann empfehle ich dir z.B. eines der anderen Kids-Bücher über C++ oder Java.)

Der Weg zum guten Programmierer kann ganz schön steinig sein. Nicht selten kommt es vor, dass man die Lust verliert, weil einfach gar nichts klappen will. Das Programm tut etwas ganz anderes, man kann den Fehler nicht finden und man fragt sich: Wozu soll ich eigentlich programmieren lernen, wo es doch schon genug Programme gibt?

Gute Programmierer werden immer gesucht, und dieser Bedarf wird weiter steigen. Und Python gehört dabei durchaus zu den erwünschten Sprachen. Wirklich gute Programmierer werden auch wirklich gut bezahlt. Es ist also nicht nur einen Versuch wert, es kann sich durchaus lohnen, das Programmieren in Python zu erlernen.

DIE ENTWICKLUNGSUMGEBUNG

Um eine Entwicklungsumgebung für Python musst du dich nicht weiter kümmern, wenn dir eine einfache reicht. Die nämlich bekommst du kostenlos mit dem Python-Paket (sie heißt IDLE = »Integrated Development and Learning Environment«). Die werden wir hier ausgiebig benutzen.

Das komplette Paket kannst du dir von dieser Seite herunterladen:

<https://www.python.org/>

Dabei muss es nicht unbedingt die neueste Version sein. Dieses Buch bezieht sich auf Python 3 (aktuelle Versionen 3.8 oder 3.9).

UND WAS BIETET DIESES BUCH?

Über eine ganze Reihe von Kapiteln verteilt lernst du

- ◇ das Basiswissen von Python kennen
- ◇ etwas über objektorientierte Programmierung
- ◇ mit Komponenten des Moduls `tkinter` zu arbeiten (das sind Bausteine, mit denen du dir viel Programmierarbeit sparen kannst)

- ◇ die grafischen Möglichkeiten von Python kennen
- ◇ etwas über den Umgang mit dem Spiele-Modul pygame
- ◇ wie man eigene Game- und Player-Klassen programmiert

Im **Anhang** gibt es dann noch zusätzliche Informationen und Hilfen, unter anderem über Installationen und den Umgang mit Fehlern.

WIE ARBEITE ICH MIT DIESEM BUCH?

Grundsätzlich besteht dieses Buch aus einer Menge Text mit vielen Abbildungen dazwischen. Natürlich habe ich mich bemüht, alles so zuzubereiten, dass daraus lauter gut verdauliche Happen werden. Damit das Ganze noch genießbarer wird, gibt es zusätzlich noch einige Symbole, die ich dir hier gern erklären möchte:

ARBEITSSCHRITTE

- Wenn du dieses Zeichen siehst, heißt das: Es gibt etwas zu tun. Damit kommen wir beim Programmieren Schritt für Schritt einem neuen Ziel immer näher.

Grundsätzlich lernt man besser, wenn man einen Programmtext selbst eintippt oder ändert. Aber nicht immer hat man große Lust dazu. Deshalb gibt es alle Projekte im Buch auch als Download:

<http://www.mitp.de/0619>

Und hinter einem Programmierschritt findest du auch den jeweiligen Namen des Projekts oder einer Datei (z.B. → PROJEKT1.PY). Wenn du also das Projekt nicht selbst erstellen willst, kannst du stattdessen diese Datei laden (zu finden im Ordner PROJEKTE).

AUFGABEN

Am Ende eines Kapitels findest du jeweils eine Reihe von Fragen und Aufgaben. Diese Übungen sind nicht immer ganz einfach, aber sie helfen dir, noch besser zu programmieren. Lösungen zu den Aufgaben findest du in verschiedenen Formaten ebenfalls im Verzeichnis PROJEKTE. Du kannst sie dir alle im Editor von Windows oder auch in deinem Textverarbeitungsprogramm anschauen. Oder du lässt sie dir ausdrucken und hast sie dann schwarz auf weiß, um sie neben deinen Computer zu legen. (Auch die Programme zu den Aufgaben liegen im Ordner PROJEKTE.)

NOTFÄLLE

Vielleicht hast du irgendetwas falsch gemacht oder etwas vergessen. Oder es wird gerade knifflig. Dann fragst du dich, was du nun tun sollst. Bei diesem Symbol findest du eine Lösungsmöglichkeit. Notfalls kannst du aber auch ganz hinten im Anhang B nachschauen, wo ein paar Hinweise zur Pannenhilfe aufgeführt sind.



WICHTIGE STELLEN IM BUCH

Hin und wieder findest du ein solch dickes Ausrufezeichen im Buch. Dann ist das eine Stelle, an der etwas besonders Wichtiges steht.



EXPERTENWISSEN

Wenn du ein solches »Wow« siehst, geht es um ausführlichere Informationen zu einem Thema.



WAS BRAUCHST DU FÜR DIESES BUCH?

Installiert wird Python mit einem Setup-Programm in ein Verzeichnis deiner Wahl, z.B. D:\PYTHON. Dort solltest du später auch deine Python-Projekte unterbringen.

Die Beispielprogramme in diesem Buch gibt es alle als Download von der Homepage des Verlages, falls du mal keine Lust zum Abtippen hast:

<http://www.mitp.de/0619>

Und auch die Lösungen zu den Fragen und Aufgaben sind dort untergebracht (alles im Ordner PROJEKTE).

BETRIEBSSYSTEM

Die meisten Computer arbeiten heute mit dem Betriebssystem Windows. Davon brauchst du eine der Versionen 7 bis 11. (Python gibt es unter anderem auch für Linux.)

SPEICHERMEDIEN

Auf jeden Fall benötigst du etwas wie einen USB-Stick oder eine SD-Card, auch wenn du deine Programme auf die Festplatte speichern willst. Auf einem externen Speicher sind deine Arbeiten auf jeden Fall zusätzlich sicher aufgehoben.

Gegebenenfalls bitte deine Eltern oder Lehrer um Hilfe.

HINWEISE FÜR LEHRER

Dieses Buch versteht sich auch als Lernwerk für den Informatik-Unterricht in der Schule. Dort setzt natürlich jeder Lehrer seine eigenen Schwerpunkte. Benutzen Sie an Ihrer Schule bereits ein Werk aus einem Schulbuchverlag, so lässt sich dieses Buch auch als Materialienband einsetzen – in Ergänzung zu dem vorhandenen Schulbuch. Weil dieses Buch sozusagen »von null« anfängt, ist ein direkter Einstieg in Python möglich – ohne irgendwelche anderen Programmierkenntnisse.

Ein wichtiger Schwerpunkt in diesem Buch ist die objektorientierte Programmierung (OOP). Auf die wichtigsten Eigenheiten (Kapselung, Vererbung und Polymorphie) wird ausführlich eingegangen. Ein anderer Schwerpunkt ist die Programmierung von Spielen. In den Projekten werden alle wesentlichen Elemente des Python-Wortschatzes wie auch die wichtigsten Grafik-Komponenten von `tkinter` eingesetzt. In den Lösungen zu den Aufgaben finden Sie weitere Vorschläge zur Programmierung.

ÜBUNGSMEDIEN

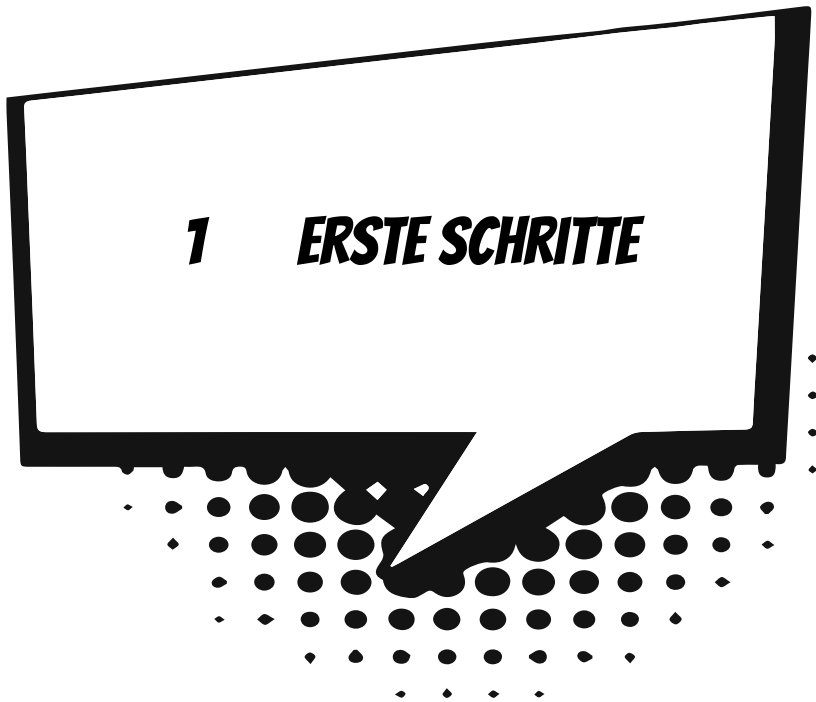
Für den Informatik-Unterricht sollte jeder Schüler ein eigenes externes Speichermedium haben, um darauf seine Programmiersuche zu sichern. So wird verhindert, dass sich auf der Festplatte des Schulcomputers mit der Zeit allerlei »Datenmüll« ansammelt. Außerdem dient der eigene Datenträger dem Datenschutz: Nur der betreffende Schüler kann seine Daten manipulieren.

AUF DIE DATEIEN ZUM BUCH VERZICHTEN?

Vielleicht ist es Ihnen lieber, wenn Ihre Schüler die Projekte alle selbst erstellen. Dann lassen Sie die Download-Dateien einfach (erst einmal) weg.

REGELMÄßIG SICHERN

Es kann nicht schaden, die Programmdateien, an denen gerade gearbeitet wird, etwa alle zehn Minuten zu speichern. Denn Computer pflegen gern gerade dann »abzustürzen«, wenn man seine Arbeit längere Zeit nicht gespeichert hat.



Hier geht es gleich ans »Eingemachte«. Nachdem wir Python installiert und gestartet haben, machen wir unsere ersten Gehversuche. Um später auch größere Programmprojekte erstellen zu können, brauchen wir das passende Werkzeug. Wir richten uns so komfortabel ein, dass schließlich auch dein erstes Programm entsteht.

In diesem Kapitel lernst du

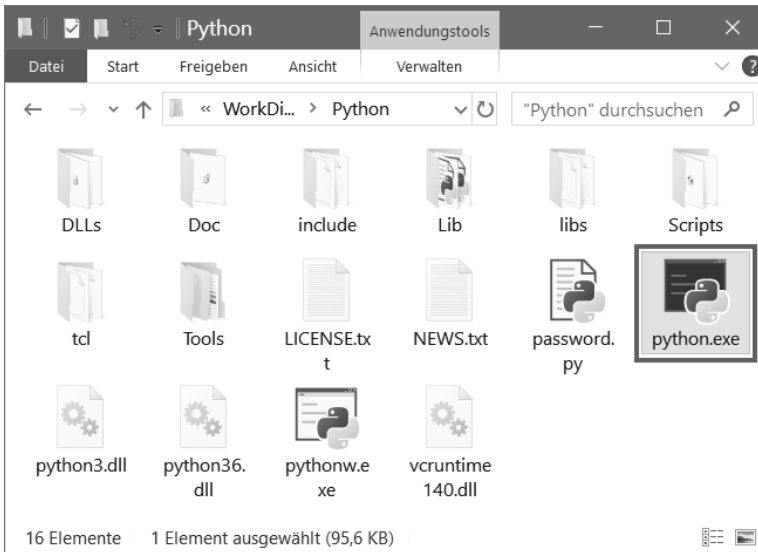
- ⊙ wie man Python startet
- ⊙ Anweisungen für Ausgabe und Eingabe kennen
- ⊙ was Variablen sind
- ⊙ den Typ String kennen
- ⊙ etwas über den Einsatz von IDLE
- ⊙ wie man ein Programm erstellt und speichert
- ⊙ wie man Python beendet

MIT PYTHON LOSLEGEN

Bevor wir mit dem Programmieren anfangen können, muss **Python** erst installiert werden.

Die Installation übernimmt ein sogenanntes Setup-Programm. Genauer erfährst du im Anhang A. Hier musst du dir von jemandem helfen lassen, wenn du dir die Installation nicht allein zutraust. Eine Möglichkeit, Python zu starten, ist diese:

➤ Öffne den Ordner, in dem du Python untergebracht hast – z.B. C:\PROGRAMME\PYTHON oder D:\PYTHON.



➤ Hier suchst du unter den vielen Symbolen das mit dem Namen PYTHON.EXE heraus. Doppelklicke auf das Symbol.

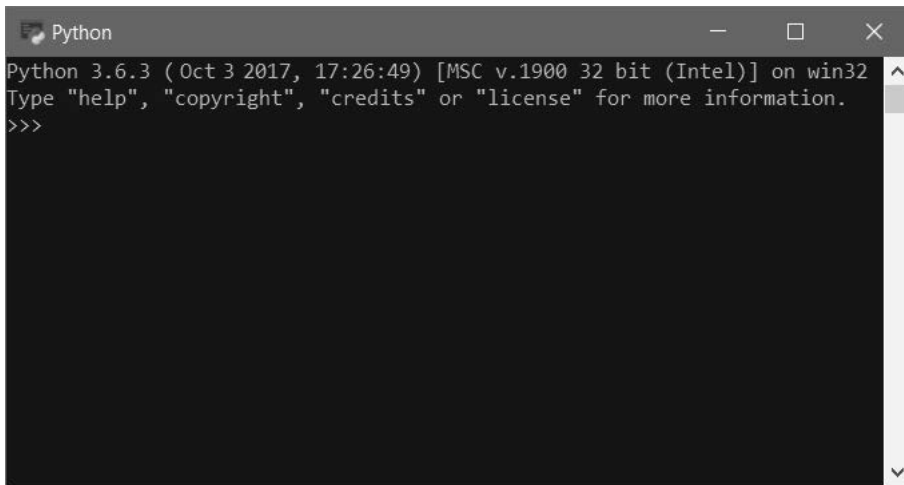


Wenn du willst, kannst du auch eine **Verknüpfung** auf dem Desktop anlegen:

- ❖ Dazu klickst du mit der rechten Maustaste auf das Symbol für Python (PYTHON.EXE). Im Kontextmenü wählst du KOPIEREN.
- ❖ Dann klicke auf eine freie Stelle auf dem Desktop, ebenfalls mit der rechten Maustaste. Im Kontextmenü wählst du VERKNÜPFUNG EINFÜGEN.
- ❖ Es ist sinnvoll, für das neue Symbol auf dem Desktop den Text python.exe – Verknüpfung einfach durch Python zu ersetzen.

Von nun an kannst du auf das neue Symbol **doppelklicken**, um die Arbeitsumgebung von Python direkt zu starten.

Was dich nach dem Start erwartet, sieht etwa so aus:



```
Python 3.6.3 (Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Die ersten beiden Zeilen informieren dich unter anderem über die aktuelle Python-Version, aber du bekommst auch schon ein paar Befehle vorgeschlagen, die du hinter den drei spitzen Klammern (`>>>`) eintippen kannst.

Die drei Zeichen werden hier auch **Prompt** genannt. Das ist eine Art Eingabeaufforderung, weil du dahinter etwas eingeben kannst (und musst, wenn es weitergehen soll).

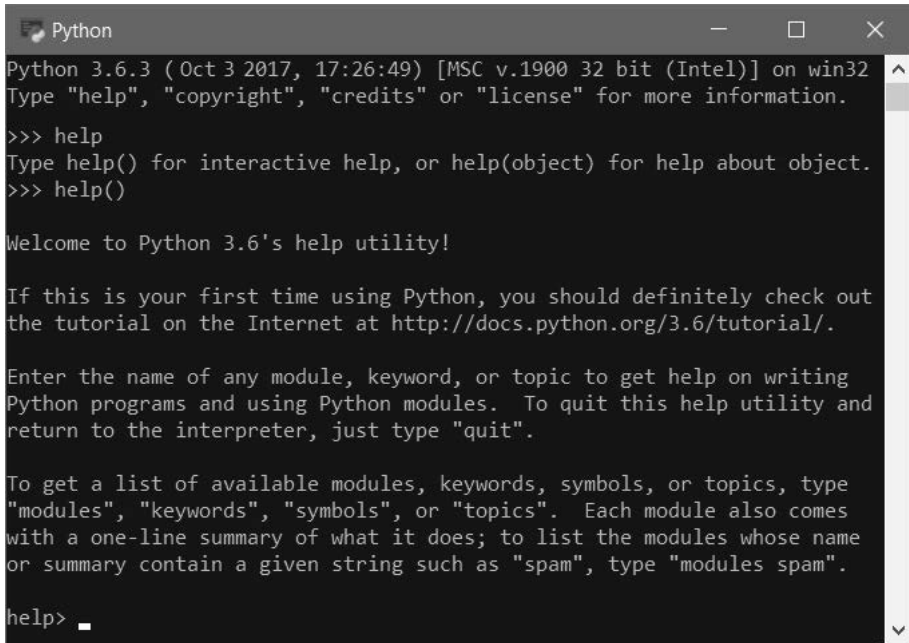


➤ Probieren wir es doch gleich einmal mit »help«. Tippe dieses Wort ein.

Und prompt gibt es etwas zu meckern: Na ja, es ist eher ein netter Hinweis: Man muss `help` mit zwei runden Klammern dahinter eintippen.

➤ Gib also `help()` ein.

Und du bekommst gleich eine ganze Menge Text serviert:

A screenshot of a Python 3.6.3 help window. The window title is "Python". The text inside shows the version and platform information: "Python 3.6.3 (Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)] on win32". It then prompts the user to type "help", "copyright", "credits" or "license" for more information. The user has entered "help", and the window displays the help text for the "help" command. The text includes: ">>> help", "Type help() for interactive help, or help(object) for help about object.", ">>> help()", "Welcome to Python 3.6's help utility!", "If this is your first time using Python, you should definitely check out the tutorial on the Internet at http://docs.python.org/3.6/tutorial/.", "Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type 'quit'.", "To get a list of available modules, keywords, symbols, or topics, type 'modules', 'keywords', 'symbols', or 'topics'. Each module also comes with a one-line summary of what it does; to list the modules whose name or summary contain a given string such as 'spam', type 'modules spam'.", and "help> _".

```
Python
Python 3.6.3 (Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> help
Type help() for interactive help, or help(object) for help about object.
>>> help()

Welcome to Python 3.6's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/3.6/tutorial/.

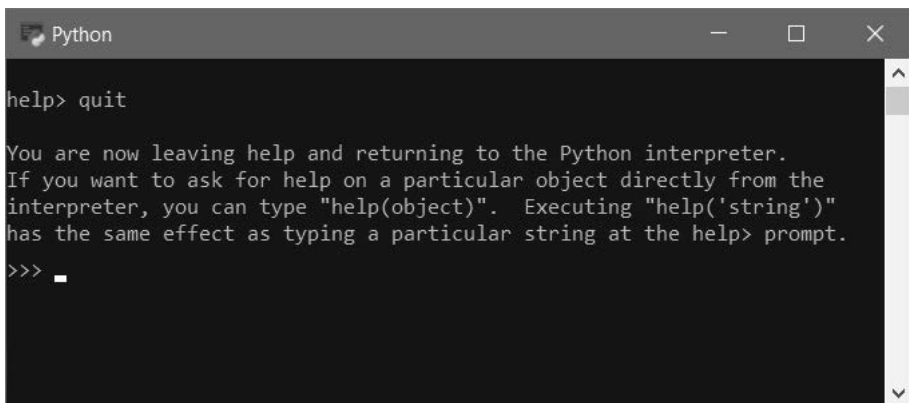
Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> _
```

Nun steht da `help>` als Prompt. Du kannst dahinter ein Wort eingeben, und wenn es zum Python-Wortschatz gehört, bekommst du dazu eine (kurze) Erläuterung.

➤ Um zum ursprünglichen Prompt zurückzukehren, tippe `quit` ein.

A screenshot of a Python 3.6.3 help window. The window title is "Python". The text inside shows the user entering "quit" at the "help>" prompt. The window then displays the message: "You are now leaving help and returning to the Python interpreter. If you want to ask for help on a particular object directly from the interpreter, you can type 'help(object)'. Executing 'help('string')' has the same effect as typing a particular string at the help> prompt." followed by ">>> _".

```
Python
help> quit

You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)". Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
>>> _
```

Und du bist wieder zurück im Python-Interpreter.

Was ist ein **Interpreter**? Zuerst solltest du wissen, dass das, was du als Befehl hinter dem Prompt eintippst, für den Computer erst einmal völlig unverständlich ist. Normalerweise kann er also den jeweiligen Befehl gar nicht ausführen. Ein Interpreter übersetzt die Befehlszeile in eine Sprache, die der Computer versteht, sodass er den Befehl ausführen kann – genannt **Maschinensprache**. Bei einem Programm, das aus einigen bis sehr vielen Zeilen bestehen kann, wird von einem Interpreter jede Zeile **einzel**n übersetzt und dann ausgeführt. Im Gegensatz dazu gibt es **Compiler**, die das **gesamte** Programm in Maschinensprache übersetzen. Erst wenn das Programm komplett und fehlerfrei ist, kann es vom Computer ausgeführt werden. Für Python benutzen wir hier einen Interpreter, es gibt aber auch Python-Compiler.



ZAHLEN UND TEXT

Nun wollen wir aber endlich mal was ausprobieren.

➤ Tippe also ein: `1+2+3` und drücke dann die `↵`-Taste.

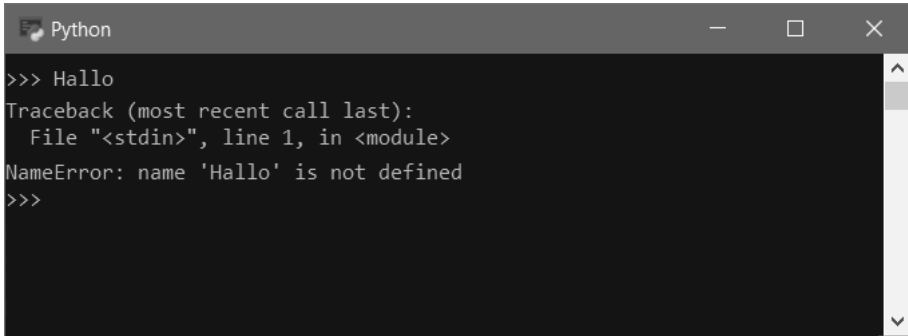
```
Python
>>> 1+2+3
6
>>>
```

Und tatsächlich wird das Ergebnis dieser kleinen Matheaufgabe angezeigt.

➤ Du kannst gern ein paar weitere Aufgaben stellen und dabei auch die Operationszeichen für minus (-), mal (*) und geteilt durch (/) benutzen.

Na ja, als Taschenrechner scheint der Python-Interpreter ja gut zu funktionieren, aber natürlich erwartest du viel mehr als das.

➤ Versuchen wir es mal mit einem netten Gruß: Tippe `Ha1lo` ein und schließe das mit der `↵`-Taste ab.

A screenshot of a Python terminal window. The title bar says "Python". The prompt is ">>>". The user has entered "Hallo". The terminal shows a "Traceback (most recent call last):" followed by "File "<stdin>", line 1, in <module>" and "NameError: name 'Hallo' is not defined". The prompt ">>>" is shown again.

```
>>> Hallo
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Hallo' is not defined
>>>
```

Na ja, irgendwie gibt es jetzt wirklich was zu meckern. »Error« heißt »Fehler«, demnach ist hier eindeutig etwas falsch. Was ist das Ziel? Ich möchte, dass der Computer ein freundliches »Hallo« sagt (bzw. schreibt).

➤ Dazu tippe jetzt mal folgende Zeile ein:

➤ `print("Hallo")`

A screenshot of a Python terminal window. The title bar says "Python". The prompt is ">>>". The user has entered "print('Hallo')". The terminal shows the output "Hallo". The prompt ">>> " is shown again.

```
>>> print("Hallo")
Hallo
>>> 
```

Das geht. Dabei bedeutet `print()` hier anzeigen, ausgeben. Und in den runden Klammern dahinter steht das, was angezeigt werden soll. Das nennt man **Parameter**.

Natürlich geht das auch mit Zahlen:

A screenshot of a Python terminal window. The title bar says "Python". The prompt is ">>>". The user has entered "print(55)". The terminal shows the output "55". The prompt ">>> " is shown again.

```
>>> print(55)
55
>>> 
```

➤ Probiere selber aus, was der `print`-Befehl alles ausgeben kann.

Nun wird es ein bisschen komplizierter. Bis jetzt haben wir immer nur einen Befehl eingegeben. Aber richtige Programme bestehen natürlich aus mehr als nur einer Zeile. Versuchen wir es mal mit diesem kleinen Programmstück:

```
Text = "Hallo"  
print(Text)
```

➤ Gib diese beiden Zeilen ein. Wird angezeigt, was du erwartet hast?



```
Python  
>>> Text = "Hallo"  
>>> print(Text)  
Hallo  
>>> _
```

Nach der ersten Zeile gibt es noch nichts anzuzeigen. Aber offenbar hat sich der Python-Interpreter gemerkt, was `Text` bedeutet. Und er weiß, welchen Wert `print()` als Parameter übernehmen soll.

Genauer: Bei `Text` handelt es sich um eine sogenannte **Variable**, der wird ein Wert zugewiesen, in diesem Fall ist das das Wörtchen "Hallo". Und das Gleichheitszeichen (=) wird **Zuweisungsoperator** genannt.

Variable = **Wert**

In Python werden Variablen neu **erzeugt**, wenn ihnen zum ersten Mal ein Wert zugewiesen werden soll. Bei einer Zuweisung steht **immer** links die Variable und rechts der Wert, das Gleichheitszeichen hat also quasi die Bedeutung eines Pfeils:

Variable ← Wert

Variablen sind nützlich, weil sie Daten »aufheben«, sodass der Computer sich an einen Wert erinnern kann. Vor allem in größeren Programmen ist es wichtig, dass der Inhalt einer Variablen auch mehrmals benutzt werden kann. Einige Beispiele dafür wirst du noch kennenlernen.



Nun hat der Computer so schön »Hallo« gesagt, das könnte man doch noch um einigen Text erweitern:

```
Text = "Hallo, wer da?"  
print(Text)  
Name = input()  
print(Name)
```



In Python muss ein Text wie unser Hallo-Gruß immer in **Anführungszeichen** gefasst werden. Ich benutze hier die doppelten ("), aber auch die einfachen (') sind erlaubt.

Diese beiden Zeilen sind also völlig gleichwertig:

```
Text = "Hallo, wer da?"  
Text = 'Hallo, wer da?'
```

➤ **Tippe** alle diese Zeilen nacheinander ein. Beachte, dass du nach der `input`-Zeile erst selber deinen Namen eingeben musst, ehe es weitergeht.

Bei mir sieht das Ganze so aus:

```
Python  
>>> Text = "Hallo, wer da?"  
>>> print(Text)  
Hallo, wer da?  
>>> Name = input()  
Hans-Georg  
>>> print(Name)  
Hans-Georg  
>>> _
```

Gar nicht so schlecht. Und dabei hast du gleich einen neuen Befehl kennengelernt: `input()` heißt hier eingeben. Damit lässt sich doch schon was anfangen. Wenn man oft genug `print()` und `input()` benutzt, kann man schon ein ansehnliches Gespräch mit dem Computer führen.

Aber etwas gefällt mir nicht. Zum Beispiel das dauernde Neueingeben. Man kann nicht einfach beliebig mit den Pfeiltasten im Programm herumwandern oder mit der Maus irgendwohin klicken und dort Text ändern. Besser wäre es doch, wenn man sich im Python-Fenster wie in einem Texteditor bewegen könnte.

Unvorstellbar, dass wir auf diese Weise größere Programmprojekte erstellen wollen. Dazu muss man auch die Möglichkeit haben, den mühsam eingetippten Text irgendwo als Datei zu speichern. Das aktuelle Werkzeug, das wir mit dem Eingabefenster haben, reicht also offenbar nicht aus.

EINE ARBEITSUMGEBUNG NAMENS IDLE

Wir brauchen also ein Fenster, über das man eingegebenen Text auch speichern und dorthin wieder laden kann. Genannt Editor. Im Python-Paket ist ein solcher Editor bereits enthalten, man muss ihn nur finden.

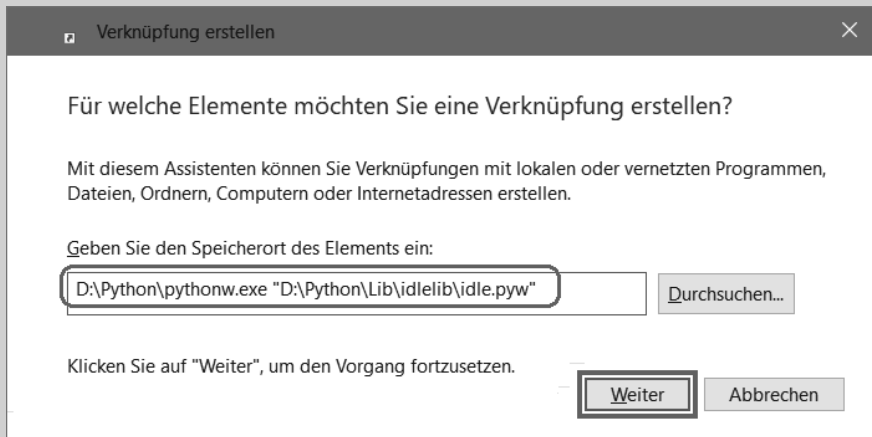
Wenn bei der Installation im START-Menü von Windows eine Verknüpfung zu Python eingerichtet wurde, dann findest du dort auch einen Eintrag wie IDLE (PYTHON).



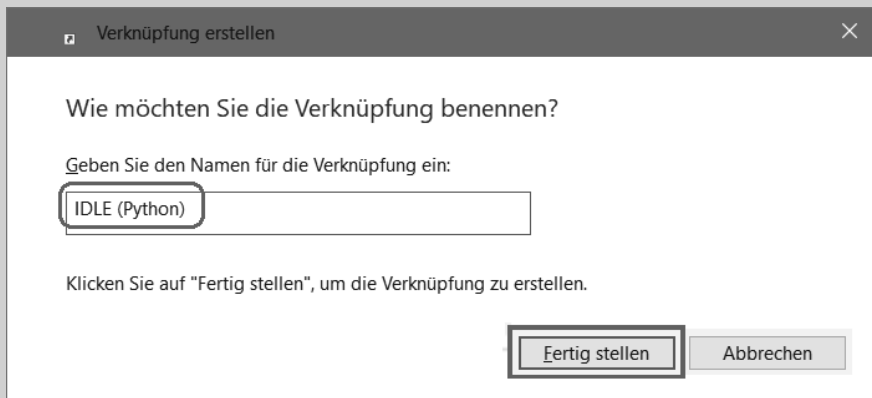
➤ Klicke darauf, um dieses Programm zu starten.

Wenn du diesen Eintrag nicht in deinem Start-Menü findest, dann kannst du dir selber eine Verknüpfung auf dem Desktop erstellen.

- ❖ Klicke mit der rechten Maustaste auf eine freie Stelle im Desktop und wähle im Kontextmenü NEU und dann VERKNÜPFUNG.
- ❖ Im Dialogfeld gibst du als SPEICHERORT DES ELEMENTS ein:
D:\Python\pythonw.exe "D:\Python\Lib\idlelib\idle.pyw"

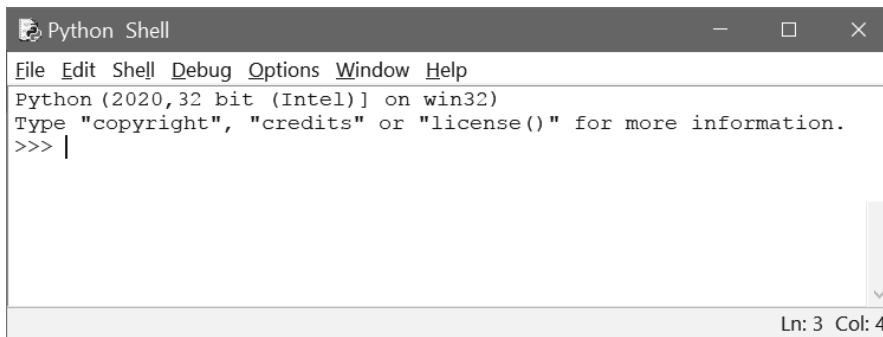


- ❖ Wichtig ist, dass D:\PYTHON auch das Verzeichnis ist, in das du Python installiert hast. Sonst musst du das entsprechend anpassen.
- ❖ Nenne das neue Symbol auf dem Desktop IDLE (Python).



Damit kannst du von nun an den Python-Editor direkt von Desktop aus per Doppelklick starten.

Nach dem Start von IDLE findest du dich in einem solchen Fenster wieder:



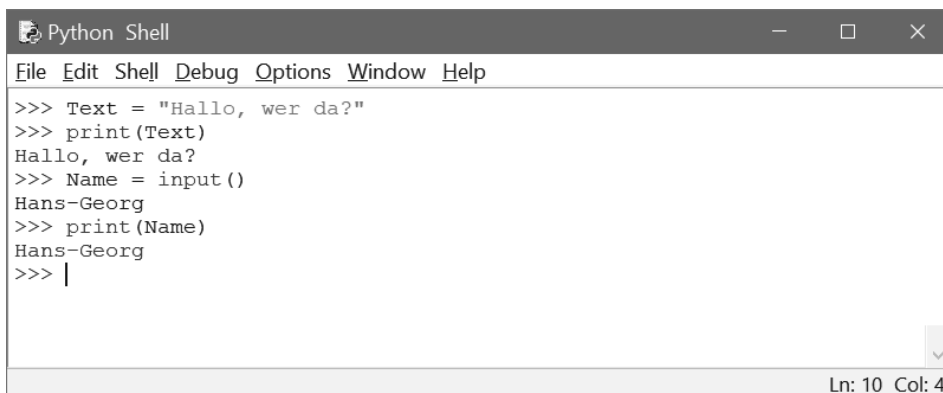
```
Python Shell
File Edit Shell Debug Options Window Help
Python (2020, 32 bit (Intel)) on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
Ln: 3 Col: 4
```

Sieht irgendwie ähnlich aus wie das Fenster des Python-Interpreters, und irgendwie auch anders. Nicht nur, dass es hier statt weiß auf schwarz umgekehrt zugeht, hier gibt es auch eine Menüleiste. Und das Ganze nennt sich »Shell«.

IDLE ist die Abkürzung für »Integrated Development and Learning Environment«, frei übersetzt ist das eine Umgebung in diesem Fall für das Entwickeln und Lernen von Python-Programmen. Der Begriff **Shell** bedeutet »Schale« und zielt in dieselbe Richtung.



Wir können auch in dieser Umgebung unsere Python-Befehle eintippen, dann bekommen wir dieselben Ergebnisse wie ganz oben (im »schwarzen« Fenster):



```
Python Shell
File Edit Shell Debug Options Window Help
>>> Text = "Hallo, wer da?"
>>> print(Text)
Hallo, wer da?
>>> Name = input()
Hans-Georg
>>> print(Name)
Hans-Georg
>>> |
Ln: 10 Col: 4
```

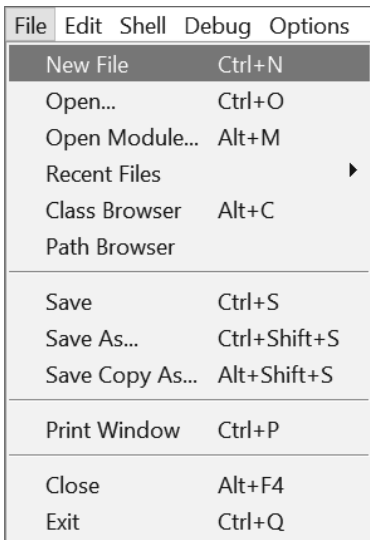
Allerdings sieht es hier etwas bunter aus. Wörter wie `print` und `input` werden farbig angezeigt, ebenso wie in Anführungsstriche gesetzter Text.

➤ Probiere das selber aus, indem du die Zeilen von oben auch hier noch mal eingibst.

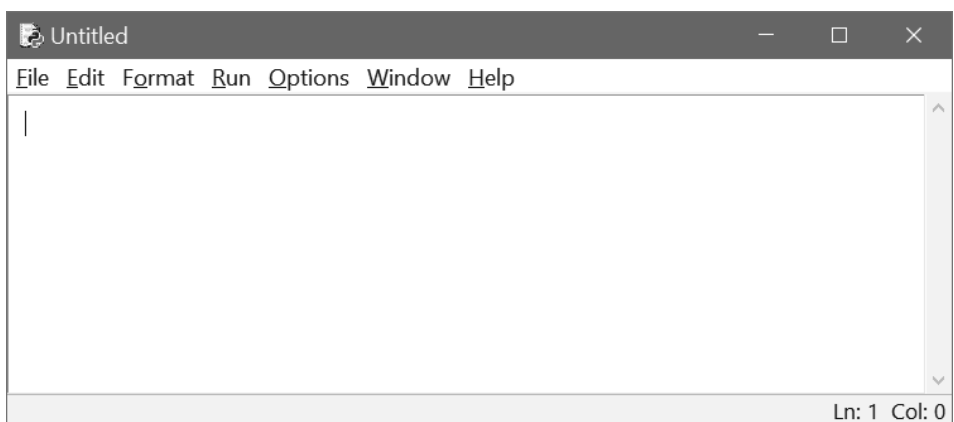
DIE ERSTE PY-DATEI

Wie kriegen wir es nun hin, dass aus den paar Zeilen ein komplettes Programm wird, das sich immer wieder laden und ausführen lässt?

➤ Klicke im Menü auf FILE und dann auf NEW FILE (oder drücke die Tastenkombination `[Strg] + [N]`).



Und schon haben wir ein weiteres (neues) Fenster mit dem Titel UNTITLED. Die Menüleiste ist ähnlich wie vorher, ansonsten ist das Fenster leer.

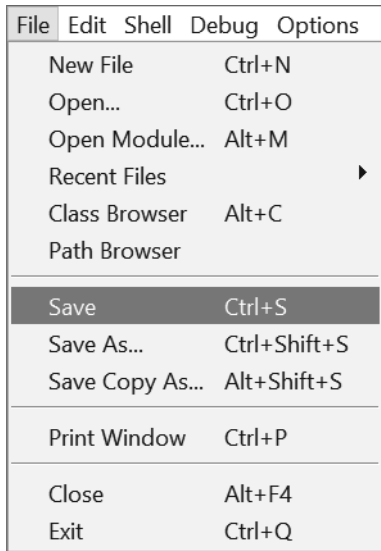


Hier können wir nun unser Programm eingeben, ohne dass irgendeine Zeile davon direkt vom Python-Interpreter ausgeführt wird.

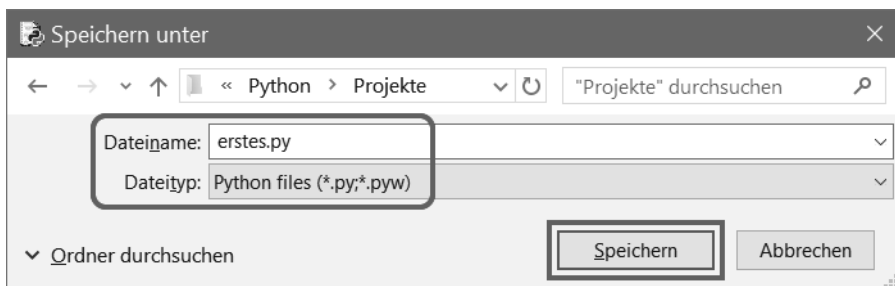
➤ Tippe also diese Zeilen ein:

```
Text = "Hallo, wer da?"  
print(Text)  
Name = input()  
print(Name)
```

➤ Und damit du sie nicht wieder verlierst (nachdem du sie ja jetzt so oft eingeben musstest), klicke nun auf FILE und SAVE oder SAVE AS.



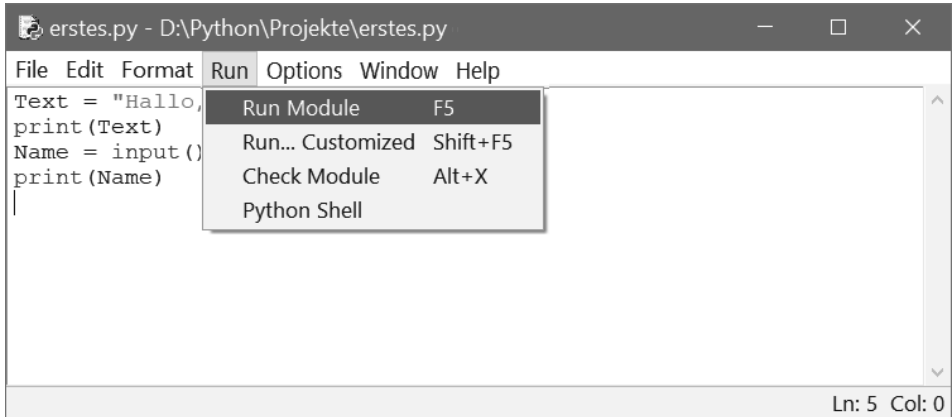
➤ Im Dialogfeld gibst du einen Namen für dieses Programm ein, z.B. erstes (oder was du willst). Das PY wird als Kennung für »Python« automatisch angehängt, wenn du es nicht angibst.



Ich habe im PYTHON-Verzeichnis einen Unterordner namens PROJEKTE erstellt und speichere meine Python-Projekte dort ab.

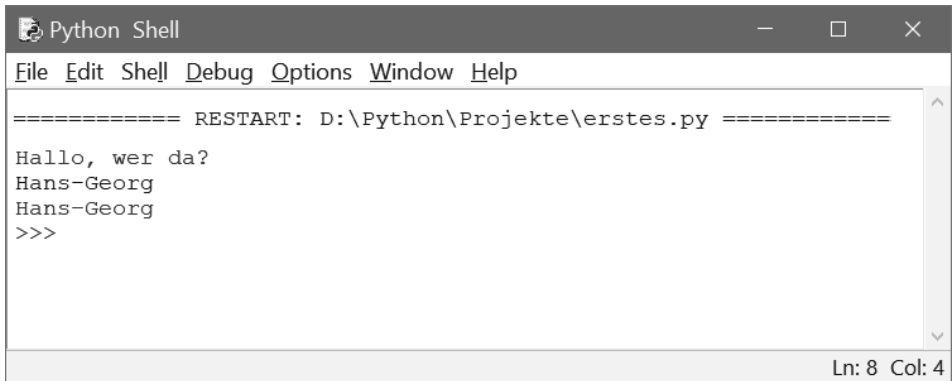


- Um das Programm zu starten, klickst du jetzt auf RUN und RUN MODULE. Oder du drückst die Taste `F5`.



Und du landest wieder im ersten Fenster, wo dich der Gruß-Text »Hallo, wer da?« erwartet.

- Tippe nun deinen Namen ein und bestätige das mit der `↵`-Taste. Dann könnte das Ergebnis so ähnlich aussehen:

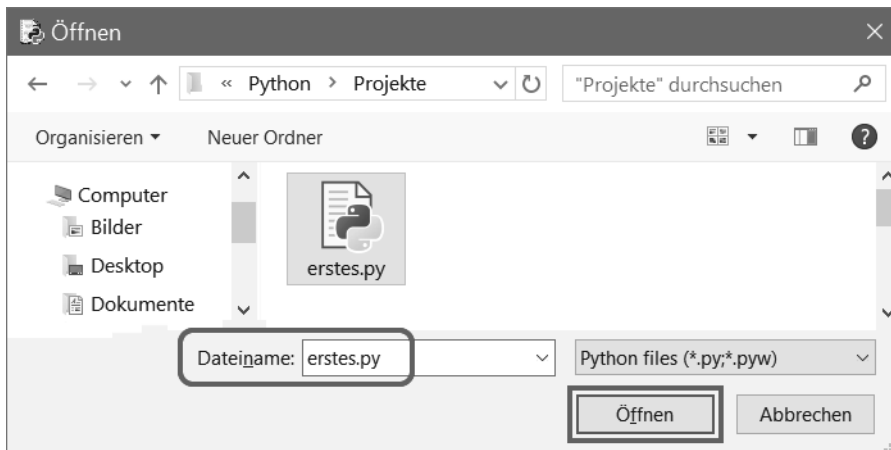


Wenn du nun dieses Fenster schließt, gibt es deine Programmdatei immer noch. Du musst sie dir einfach nur wieder zurückholen.

- Dazu klickst du auf FILE und OPEN.

File	Edit	Shell	Debug	Options
New File	Ctrl+N			
Open...	Ctrl+O			
Open Module...	Alt+M			
Recent Files	▶			
Class Browser	Alt+C			
Path Browser				
Save	Ctrl+S			
Save As...	Ctrl+Shift+S			
Save Copy As...	Alt+Shift+S			
Print Window	Ctrl+P			
Close	Alt+F4			
Exit	Ctrl+Q			

- Im Dialogfeld wählst du die betreffende Datei (bei mir im Ordner PYTHON\PROJEKTE).



- Klicke auf ÖFFNEN, das Editor-Fenster öffnet sich und dein erstes Programm ist wieder verfügbar.

In Python wird ein Programm in einer Datei auch Skript genannt.



QUELLTEXT-SPIELEREIEN

Schauen wir uns jetzt diesen Quelltext, wie man die Summe der Textzeilen auch nennt, noch einmal näher an:

```
Text = "Hallo, wer da?"
print(Text)
Name = input()
print(Name)
```

Es gibt zwei Variablen `Text` und `Name`. In denen wird jeweils eine sogenannte Zeichenkette gespeichert, auch **String** genannt. Außerdem werden hier zwei Funktionen benutzt:

<code>print()</code>	Ausgabe von Zahlen und Strings auf dem Monitor
<code>input()</code>	Eingabe von Zahlen und Strings über die Tastatur

Wie du siehst, haben Funktionen immer runde Klammern als »Anhängsel«, in denen kann etwas drinstehen, sie können aber auch leer sein – je nach Art und Anwendung der Funktion.

Und wenn du noch etwas genauer hinschaust, dann fällt dir auf, dass `print()` direkt als Anweisung aufgeführt wird, `input()` aber wird in einer **Zuweisung** eingesetzt. Also wird über diese Funktion ein Wert an die Variable `Name` zugewiesen.

Variable = Formel

Dass ich hier die Bezeichnung »Formel« benutze, soll bedeuten, dass man auf der rechten Seite der Zuweisung außer Funktionen auch z.B. so etwas einsetzen kann:

```
Zahl = 1 + 2 * 3
Text = "Du bist also " + Name
```



Wobei das Plus (+) offenbar eine Doppelrolle hat: Man kann damit Zahlen addieren und Strings verketteten.

Nachdem du jetzt weißt, wie man auch eine ganze Reihe von Programmzeilen sammeln und als Datei speichern kann, setzen wir doch gleich mal unser erstes Beispiel fort:

```
Text = "Hallo, wer da?"
print(Text)
Name = input()
Text = "Du bist also " + Name
print(Text)
print("Und wie geht es?")
Antwort = input()
print("Dir geht es also " + Antwort);
```

Wie du siehst, habe ich nicht überall Variablen benutzt, eigentlich ist das nur nötig, wenn man will, dass der Computer sich etwas merkt. Das betrifft in unserem Beispiel nur die Eingaben. Demnach kann unser Programmprojekt auch so aussehen:

```
print("Hallo, wer da?")
Name = input()
print("Du bist also " + Name)
print("Und wie geht es?")
Antwort = input()
print("Dir geht es also " + Antwort);
```

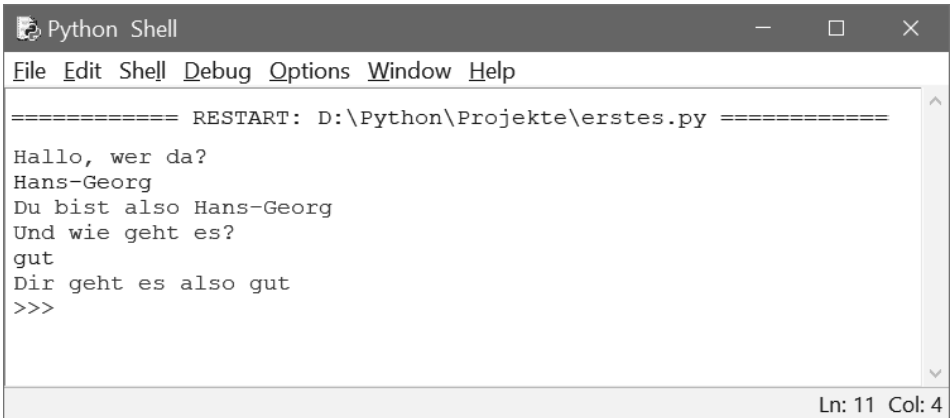
➤ Tippe erst den oberen Quelltext ein, probiere ihn über RUN und RUN MODULE (oder **F5**) aus. Dann ändere alles so, dass daraus die zweite Version wird. Lasse das Programm erneut laufen.

Bei jeder Änderung wirst du vor dem Programmstart aufgefordert, den Quelltext zu speichern:



➤ Klicke dann auf OK.

Und so könnte unsere letzte Version im Python-Interpreter-Fenster ablaufen:



```

Python Shell
File Edit Shell Debug Options Window Help

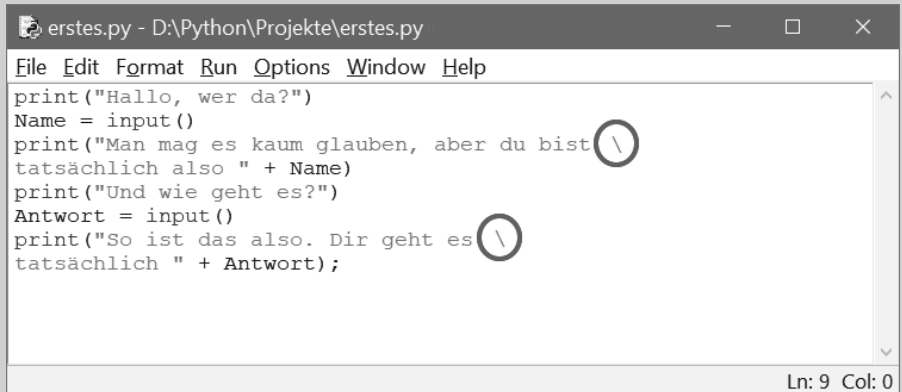
===== RESTART: D:\Python\Projekte\erstes.py =====

Hallo, wer da?
Hans-Georg
Du bist also Hans-Georg
Und wie geht es?
gut
Dir geht es also gut
>>>

```

Ln: 11 Col: 4

Wie du sehen kannst, ist Python zeilenorientiert: In **jeder** Zeile steht eine Anweisung (auch eine Zuweisung ist eine Anweisung). Man darf also Anweisungen nicht einfach auf zwei Zeilen verteilen. Und wenn das doch mal nötig sein sollte, muss am Ende der ersten Zeile ein sogenannter Backslash stehen, das ist ein umgekehrter Schrägstrich (`\`).

```

erstes.py - D:\Python\Projekte\erstes.py
File Edit Format Run Options Window Help

print("Hallo, wer da?")
Name = input()
print("Man mag es kaum glauben, aber du bist \
tatsächlich also " + Name)
print("Und wie geht es?")
Antwort = input()
print("So ist das also. Dir geht es \
tatsächlich " + Antwort);

```

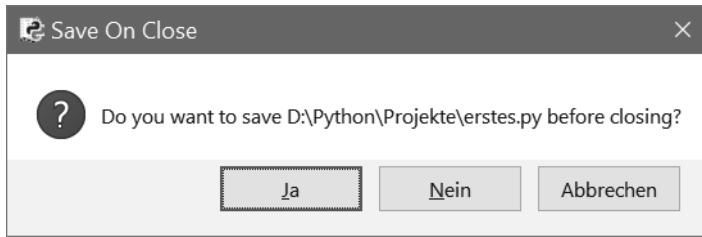
Ln: 9 Col: 0

PYTHON VERLASSEN

Um die Python-Umgebung zu beenden, müssen alle offenen Fenster geschlossen werden:

- Das geht entweder über das jeweilige FILE-Menü und den Eintrag EXIT. Oder du klickst auf das kleine X oben rechts in der Titelleiste. Wenn du willst, kannst du auch die Tastenkombination `[Strg]+[Q]` benutzen.

Solltest du vorher noch etwas am Text geändert haben, dann erscheint zuerst dieses Meldefenster:



➤ Je nach Lage klickst du auf einen der Buttons. (Mit ABBRECHEN kehrst du zum Editor zurück.)

ZUSAMMENFASSUNG

Mit deinem ersten Python-Projekt hast du schon einiges geschafft. Mal sehen, was du von diesem Kapitel behalten hast. Da wären zuerst mal ein paar Operationen im Umgang mit der Python-Shell (IDLE):

IDLE starten	Doppelklicke auf das Symbol für IDLE (Python)
Datei speichern	Klicke auf FILE/SAVE
Datei neu speichern	Klicke auf FILE/SAVE AS
Datei öffnen/laden	Klicke auf FILE/OPEN
Programmprojekt starten	Klicke auf RUN/RUN MODULE
Hilfesystem aufrufen	help() eintippen (mit quit verlassen)
Python beenden	Klicke auf FILE/EXIT

Und ein klein wenig vom Python-Wortschatz hast du auch schon kennengelernt:

print()	Funktion für die Anzeige von Zahlen und Text
input()	Funktion für die Eingabe von Zahlen und Text
=	Zuweisungsoperator
+	Operator für Addition oder Verkettung
\	Zeilen-Trenner

EIN PAAR FRAGEN ...

1. Welche Funktionen sind für Eingabe und Ausgabe zuständig?
2. Was ist der Unterschied zwischen Compiler und Interpreter?
3. Ist eine Zuweisung dasselbe wie eine Gleichung?

... ABER NOCH KEINE AUFGABE

STICHWORTVERZEICHNIS

?= 82

#-Zeichen 47

+= 80

A

accelerator 195

add_cascade 184

add_command 185

Addition 47

after 221

anchor 155

and 61

Anführungszeichen 24

Animation 231

Anweisungsblock 41

append 84, 148

as 246

askokcancel 194

askopenfilename 187

asksaveasfilename 188

askyesno 193

atan2 265

Attribut 111

Auflösung 200

Ausschneiden 60

B

background 210

Backslash 34, 79, 174, 209

Bedingung 41

Beenden

 Programm 69

bg 210

Bildpunkt 200

bind 151, 192, 212

blit 302

Bogenmaß 265

Boolesche Variable 85

border 160

break 70

bubblesort 104

Button 134

 command 135

Button-1 192

Button-3 192

buttonClick 135, 149, 170

C

Canvas 203, 217

Checkbutton 156

circle 216, 243

class 111, 142

close 175

column 140

command 135, 186

Compiler 12, 21

config 137, 139

continue 71

coords 237

create_image 223

create_line 204

create_oval 204

create_rectangle 204

create_text 209

curselection 152

D

Datei

 Neu 46, 58

Datei öffnen 174, 179

Datei schließen 175

Daten lesen 174

Daten schreiben 179

def 93

degrees 265
 delete 171, 222
 destroy 189
 Dezimalzahl 49
 display 240
 Division 47
 Dodging 293
 draw 243
 Dropdown-Menü 191

E

Editor 12, 25
 Einfügen 60
 Eingabeaufforderung 19, 323
 Eingabefeld 168
 Einrückung 39
 Element
 öffentlich 126
 privat 126
 elif 45
 ellipse 245
 else 42
 end 60
 end= 61
 Endlos-Schleife 81, 82
 Entry 168
 Entwicklungsumgebung 12
 Ereignis 134
 Motion 212
 Error 39
 Event 134
 event 152, 241, 250
 x_root, y_root 192
 event.type 241, 250
 except 54
 Exception 52
 exit 70

F

False 85
 Fehler 39
 Fehler abfangen 52
 Fenster
 schließen 46
 Titel 278

file 223
 filedialog 187
 filetypes 187
 fill 205, 243
 flip 254
 float 50
 Font 300
 font 210
 for 77, 104
 forward 214
 Frame 160, 166
 from 124
 Funktion 32
 Funktionskopf 93
 Funktionsrumpf 93

G

Game-Klasse 303
 get 154
 get_pos 262
 get_ticks 287
 getTime 287
 gif 223
 global 96
 Globale Variable 95
 goto 216
 Gradmaß 265
 Grafik
 Ellipse 204
 Farbe 205
 Linie 204
 move 221
 Rechteck 204, 207
 Text 209
 Vieleck 208
 Grafikkarte 200
 Grenzen 257
 grid 140
 Grundrechenarten 47

H

height 139
 hidelImage 222
 home 215

I

IDLE 25
if 39
image 223, 246
image.load 247
import 63, 75, 122
 * 124
import-as 246
in 77
__init__ 115, 143
init 115, 143, 228
initGame 94
Initialisierung 115
 Startwerte 94
input 24
insert 151
Installation 315
Instanz 111
int 49
Interpreter 12, 21
IntVar 154
isKilled 275
itemconfig 232

K

Kapselung 110
key.get_pressed 253
Keydown 250
keysym 195
Klammern
 rund 59
Klasse 110
 Datei 120, 228, 275
Kollision 274, 293
Kommentar 47
Komponente 132
Konstruktor 115
Kontakt 293
Kontextmenü 192
Kontrollfeld 156
Kontrollstruktur 41, 43, 53, 65
Kopieren 60

L

Label 133
 config 137
lambda 149, 231
Layout 138
Leerkette 171
Leerzeile 76
left 214
len 87
line 245
Listbox 151
listboxSelect 151
Liste 77
 append 84
 leer 84
 sort 106
load 247
Lokale Variable 95

M

mainloop 134
Maschinensprache 21
math
 atan 265
 degrees 265
 sqrt 265
Maus
 Motion 212
Meldebox 141
Menu 184
messagebox 141, 193
Methode 111
minsize 155
Modul
 Zeit 75
 Zufall 62
Motion 212
mouse.get_pos 262
Mousebuttondown 262
mouseDraw 212
move 232, 266
moveImage 221
Multiplikation 47

N

not 86

O

object 117

Objektorientierte Programmierung 110

öffentlich 127

open 174, 179, 188

openFile 187

Operator

 ?= 82

 != 44

 += 80

 == 39

 and 61, 271

 gleich 62

 größer 62

 kleiner 62

 or 61, 271

 Rechnen 47

 ungleich 62

 Zuweisung 23

Optionale Parameter 250

Optionsfeld 153

or 61

orient 168

P

pack 133

padx, pady 140

paint.net 247

Palindrom 87

Parameter 22, 63, 97

 optional 250

pass 111, 169, 220

Pause 75

pendown 216

penup 216

Photolmage 223

Pillow 223

Pixel 200

place 138

Player 228, 275

Player-Klasse 245

playGame 94

PNG 247

Popup-Menü 191

post 192

print 22

privat 127

Programmieren 11

Programmiersprache 12

Programmierung
 objektorientiert 110

Programm-Modul 120

Prompt 19

protocol 194

Pygame 240

 Bewegung 250

 blit 248

 Drehung 254

 import 240

 installieren 319, 323

 load 247

 quit 240

 sprite 246

 Tastatur 250

Pythagoras 265

Python 11

 beenden 34

 installieren 315

 starten 18

Q

Quadratwurzel 265

Quelltext 32

R

Radiobutton 153

randint 63, 269

random 62, 63, 269

range 77

read 179

rect 245

Referenz 106

relief 160, 167

render 300

Rendern 244, 300

return 99

 mehrfach 102

rotate 254

row 140

rstrip 174

S

saveFile 187
Scale 168
scale 254
scaleValue 172
Schalt-Variable 85, 241
Schieberegler 168
Schleife 66
Schlüsselwörter 55, 92
Schrittweite 77
Screen 214
SDL 240
self 113, 143
set 154
set_caption 278
set_mode 240
setPosition 290
set_repeat 252
setState 285
Setup 18
Shell 27
Shortcut 194
showImage 221, 231
showinfo 141, 193
showvalue 169
side 138
Skript 31
sleep 75
sort 106
Sortieren 104
Spielfeld 245
Spielfigur 245
Spirale 215
Sprite 245
 Bewegung schräg 253
 Position 248
sqrt 265
Stern 124
sticky 140, 159
str 69, 94, 227
String 32
 len 87
 verketten 32
Subtraktion 47
super 247
Surface 243, 300
SysFont 300

T

Tauschen 101
tearoff 186
Text
 Font 210
Textur 245
time 75
time.delay 267
time.get_ticks 287
title 141
Tk 133
tkinter 132
 pack 138
 place 138
transform 254
Transformation 256
Transparenz 247, 249
True 85
try 53
turnImage 233
Turtle
 Grafik 214
 home 215
 Screen 214
Typumwandlung 49, 50, 69

U

Unterstriche 126, 127
update 221, 244
Ursprung 200

V

value 153
Variable 23
 erzeugen 65
 global 95
 lokal 95
variable 153
Vektor 262
Vererbung 116
Vergleichsoperator 39, 62
Verknüpfungsoperator 61
virtuell 120

W

Warten 75
while 64, 80
Widgets 132
width 139
Wiederholung 64
Winkel 254
Winkelfunktion 265
WM_DELETE_WINDOW 194
Wortschatz 92
write 179

X

x-Achse 200
x_root 192

Y

y-Achse 201
y_root 192

Z

z-Achse 202
Zählschleife 77
Zeichenkette 32
Zeilentrennung 79
Zeilenumbruch 61
Zeit-Modul 75
Zufall 269
Zufalls-Modul 62
Zugriff
 öffentlich 127
 privat 127
Zuweisung 67
 mehrfach 103
Zuweisungsoperator 23, 39
Zwischenablage 60