



Programmieren in Python für den Grafiktaschenrechner TI-84 Plus CE-T *Python Edition*

Version 5.7.0 Bundle 84CE-T

Verwenden Sie die Online-Hilfe unter education.ti.com/eguide, um mehr über die TI-Technologie zu erfahren.

Wichtige Informationen

Sofern in der jeweiligen Programmlizenz nicht ausdrücklich anders aufgeführt, übernimmt Texas Instruments für die Programme oder das Handbuchmaterial keinerlei Garantie, weder direkt noch indirekt. Dies umfasst auch jegliche indirekte Gewährleistung hinsichtlich der Marktgängigkeit oder der Eignung für einen bestimmten Zweck, ist jedoch nicht hierauf beschränkt und dieses Produkt wird lediglich „so wie es ist“ zur Verfügung gestellt. In keinem Fall kann Texas Instruments für Schäden haftbar gemacht werden, die sich entweder in Verbindung mit dem Kauf bzw. Gebrauch dieses Produkts ergeben oder dadurch verursacht werden, dies gilt für spezielle, begleitende und versehentliche Schäden sowie für Folgeschäden. Texas Instruments haftet maximal und ausschließlich in der Programmlizenz festgelegten Höhe, unabhängig vom jeweiligen Fall. Weiterhin haftet Texas Instruments nicht für Forderungen einer anderen Partei, die sich aus dem Gebrauch dieses Produkts ergeben, welcher Art diese Forderungen auch immer sein mögen.

„Python“ und die Python-Logos sind Warenzeichen oder eingetragene Warenzeichen der Python Software Foundation, die von Texas Instruments Incorporated mit Erlaubnis der Foundation verwendet werden.

Hinweis: Die tatsächlichen Bildschirme können leicht von den bereitgestellten Bildern abweichen.

© 2019 - 2021 Texas Instruments Incorporated

Inhalt

Neuerungen	1
Neuerungen in der Python-App	1
Python-App	2
Verwenden der Python-App	3
Speicherverwaltung für Python-Programme (PY AppVar)	3
Navigation in der Python-App	4
Beispielaktivität	5
Einrichten einer Python-Sitzung mit Ihren Programmen	7
Python-Arbeitsbereiche	8
Python-Dateimanager	9
Python-Editor	11
Python-Shell	14
Unterstützung der Schnellbearbeitung	18
Verwenden der Python-Tastatur	18
Verwenden des Python-Katalogs	20
Verwenden der [a A #] Zeichentabelle	21
[Fns...] Menüs, Module und Add-on-Module	22
[Fns...] Menüs	22
[Fns...] Integriert, Operatoren und Schlüsselwörter	22
[Fns...] Module	22
[Fns...] Add-on-Module	28
[Fns...] ti_draw Add-on-Modul	29
[Fns...] ti_image Add-on-Modul	35
Meldungen in der Python-App	39
Verwendung von TI-SmartView™ CE-T und der Python-Umgebung	41
Verwenden von TI Connect™ CE zum Konvertieren von Python-Programmen	43
Was ist die Python-Programmierungsumgebung?	44
Im TI-84 Plus CE-T Python Edition enthaltene Module	44
Beispielprogramme	51
Referenz-Leitfaden für die TI-Python Umgebung	59
KATALOG-Liste	59
Alphabetische Auflistung	59

Anhang	152
Ausgewählte Inhalte der in TI-Python integrierten Funktionen, Schlüsselwörter und Module	153
Tastaturbelegung für wait_key()	167
Allgemeine Informationen	168
Online-Hilfe	168
Kontakt mit TI Support aufnehmen	168

Neuerungen

Neuerungen in der Python-App

TI-84 Plus CE-T Python Edition

Python-Programmierung

- Bei geladener Python-App können Sie über [prgm] auf die Python-App zugreifen. Die Python-App ist auch unter [2nd] [apps] aufgelistet.
 - Halten Sie sich auf dem Laufenden: education.ti.com/84cetupdate.
 - Details zur Python-App erhalten Sie in der Python-Programmieranleitung unter education.ti.com/eguide.
- Importanweisungen für Zusatzmodule lassen sich schnell einfügen. Add-on-Module sind über die Beiträge zu Python erhältlich: education.ti.com.
- Die neuen Add-on-Module `ti_draw` und `ti_image` werden mit dem CE-Bundle heruntergeladen.
 - Zeichnen und verwenden Sie Bilder in Ihren Python-Programmen.
- Das Modulmenü von `ti_system` verwendet nun zur leichteren Bedienbarkeit die `wait_key()`-Methode.
- Die Module `ti_hub` und `ti_rover` unterstützen den neuesten TI-Innovator™ Hub sketch v 1.5.
 - Datenerfassung – mehrere Datensamples mit einem einzigen Befehl erfassen
 - Zusammengesetzte Anweisungen – mehrere Outputs synchronisieren
 - TI-RGB Array – mehrere LEDs steuern
 - Töne – mit einem einzigen Befehl wiederholte Signaltöne abspielen
 - Ranger – Laufzeit ausgeben

Übertragen von Python-Programmen

Beim Übertragen von Python-Programmen von einer Nicht-TI-Plattform auf eine TI-Plattform ODER von einem TI-Produkt auf ein anderes:

- Python-Programme, die Kernsprachenfunktionen und Standardbibliotheken (`math`, `random` usw.) verwenden, können ohne Änderungen importiert oder exportiert werden.
Hinweis: Die Begrenzung der Listenlänge beträgt 100 Elemente.
- Programme, die plattformspezifische Bibliotheken – `matplotlib` (für PC), `ti_plotlib/ti_system/ti_hub/usw.` für TI-Plattformen verwenden, müssen bearbeitet werden, bevor sie auf einer anderen Plattform ausgeführt werden können.

Dies kann sogar zwischen TI-Plattformen erforderlich sein.

Weitere Informationen zu den neuen und aktualisierten Funktionen finden Sie unter education.ti.com/84cetupdate.

Python-App

Informationen zur Verwendung, Navigation und Ausführung der Python-App finden Sie unter folgenden Punkten.

- [Verwenden der Python-App](#)
 - [Speicherverwaltung für Python-Programme \(PY AppVar\)](#)
- [Navigation in der Python-App](#)
- [Beispielaktivität](#)
- [Einrichten einer Python-Sitzung mit Ihren Programmen](#)

Verwenden der Python-App

Die Python-App ist für den TI-84 Plus CE-T *Python Edition* erhältlich. Die Informationen in diesem eGuide sind für die Verwendung mit dem I-84 Plus CE-T *Python Edition* mit dem neuesten CE-Bundle bestimmt.

Wenn Sie die Python-App zum ersten Mal auf Ihrem I-84 Plus CE-T *Python Edition* ausführen, weist Sie die App möglicherweise an, auf das neueste CE-Bundle für die neueste Python-App zu aktualisieren.

Gehen Sie auf education.ti.com/84cetupdate, um Ihren TI-84 Plus CE-T *Python Edition* zu aktualisieren.

Speicherverwaltung für Python-Programme (PY AppVar)

Die Python-App bietet einen Dateimanager, einen Editor zum Erstellen von Programmen und eine Shell, um die Programme auszuführen und mit dem Python-Interpreter zu interagieren. Python-Programme, die als Python AppVars gespeichert oder erstellt werden, werden aus dem RAM ausgeführt. Python AppVars können zur Unterstützung der Speicherverwaltung über **[2nd]** **[mem]** 2: archiviert werden. Wird eines Ihrer **PY AppVar**-Programme im Dateimanager-Bildschirm „File Manager“ der Python-App nicht angezeigt, so können Sie ein **PY AppVar**-Python-Taschenrechnerprogramm wie abgebildet zwischen dem RAM oder dem Archivspeicher verschieben. Das * markiert Dateien, die sich im Archivspeicher befinden. Drücken Sie **[enter]**, um eine Datei zwischen dem RAM und dem Archiv zu verschieben.



Hinweis: Wenn es sich bei Ihrem Taschenrechner um den TI-84 Plus CE-T *Python Edition* handelt, finden Sie unter education.ti.com/84cetupdate die neuesten Informationen für Ihren CE.

Navigation in der Python-App

Verwenden Sie in der App die Schnell Tasten auf dem Bildschirm, um zwischen den Arbeitsbereichen in der Python-App zu navigieren. In der Abbildung zeigen die Beschriftungen der Verknüpfungs-Registerkarten Folgendes an:

- * Navigation zum [Dateimanager](#) [Files]
- ** Navigation zum [Editor](#) [Edit] oder [Editor]
- *** Navigation zur [Shell](#) [Shell]

Auf die Verknüpfungs-Registerkarten im Bildschirm greifen Sie über die Grafik-Tastenzeile direkt unter dem Bildschirm zu. Siehe auch [Tastatur](#). Die Menüs [Editor>Tools](#) und [Shell>Tools](#) enthalten ebenfalls Navigationsaktionen.



Beispielaktivität

Nutzen Sie die Beispielaktivität, um sich mit den Arbeitsbereichen in der Python-App vertraut zu machen.

- Erstellen Sie ein neues Programm über den [Dateimanager](#).
- Schreiben Sie das Programm im [Editor](#).
- Führen Sie das Programm in der [Shell](#) der Python-App aus.

Weitere Informationen zur Python-Programmierung auf Ihrem CE finden Sie in den Ressourcen für den TI-84 Plus CE-T *Python Edition*.

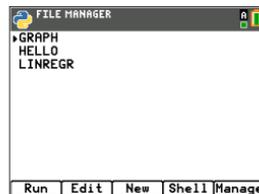
Erste Schritte:

- Führen Sie die Python-App aus.

Hinweis: Die tatsächlichen Bildschirme können leicht von den bereitgestellten Bildern abweichen.

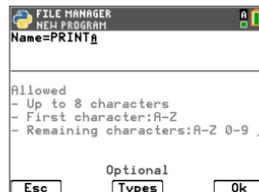
Geben Sie über den Dateimanager einen neuen Programmnamen ein.

- Drücken Sie `zoom` ([New]), um ein neues Programm zu erstellen.



Eingabe des neuen Dateinamens

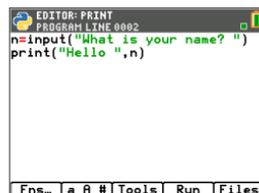
- Das Beispielprogramm wird „PRINT“ genannt. Geben Sie den Programmnamen ein und drücken Sie `graph` ([Ok]).
- Beachten Sie, dass sich der Cursor im ALPHA-Sperrmodus befindet. Geben Sie immer einen Programmnamen nach den auf dem Bildschirm vorgegebenen Vorgaben ein.



Tipp: Wenn sich der Cursor nicht im ALPHA-Sperrmodus befindet, drücken Sie `2nd` `alpha` `alpha` für Großbuchstaben.

Geben Sie das Programm wie gezeigt ein.

Tipp: Die App ermöglicht eine schnelle Eingabe! Achten Sie während der Eingabe Ihres Programms immer auf den Cursor-Zustand!



Alphabetische Zeichen auf der Tastatur	<code>alpha</code> schaltet den Status des Einfüge-Cursors in Editor und
--	--

	Shell um. _ nicht alphabetisch a Kleinbuchstabe A Großbuchstabe
Wo ist das Gleichheitszeichen?	Drücken Sie [sto→] , wenn der Cursor auf _ steht. 
Wo befinden sich input() print()	[Fns...] I/O 1:print() 2:input()
Wo befindet sich das doppelte Anführungszeichen?	[alpha] ["] 
Wo befinden sich (und)?	Benutzen Sie die Tastatur, wenn der Cursor auf _ steht. 

Try-It! [\[a A #\]](#) und [\[2nd\]](#) [\[catalog\]](#) sind bei Bedarf ebenfalls Helfer für die schnelle Eingabe!

Führen Sie das Programm PRINT aus

- Drücken Sie im Editor **[trace]** (**[Run]**), um Ihr Programm in der Shell auszuführen.
- Geben Sie Ihren Namen bei der Eingabeaufforderung „What is your name?“ ein.
- In der Ausgabe wird „HELLO“ mit Ihrem Namen angezeigt.

Hinweis: An der Shell-Eingabeaufforderung `>>>` können Sie einen Befehl ausführen, wie z. B. `2+3`. Wenn Sie eine Methode aus `math`, `random` oder anderen verfügbaren Modulen verwenden, müssen Sie wie in jeder Python-Codierungsumgebung zuerst eine Modul-Importanweisung ausführen.

Shell-Cursor-Statusanzeige.

Geben Sie Ihren Namen ein.
Ausgabe von PRINT wird angezeigt.



Einrichten einer Python-Sitzung mit Ihren Programmen

Wenn die Python-Anwendung gestartet wird, wird die CE-Verbindung für Ihre aktuelle Python-Sitzung mit der TI-Python-Umgebung synchronisiert. Sie sehen die Liste Ihrer Programme im RAM und in den dynamischen Modulen, während diese sich mit der Python-Umgebung synchronisieren.

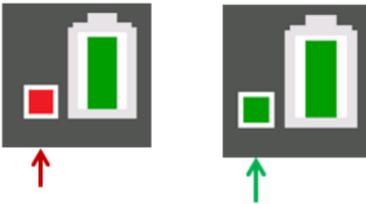
Wenn die Python-Sitzung eingerichtet ist, signalisiert die Statusleiste durch eine grüne quadratische Anzeige neben dem Batteriesymbol, dass die Python-Sitzung einsatzbereit ist. Falls die Anzeige rot ist, warten Sie, bis sie wieder auf grün wechselt, nachdem die Python-Umgebung wieder verfügbar ist.

Möglicherweise sehen Sie eine Aktualisierung der Python-Distribution, wenn Sie die Python-App zusammen mit der Programmsynchronisierung nach der letzten Aktualisierung Ihres

TI-84 Plus CE-T *Python Edition* über education.ti.com/84cetupdate starten.

Trennen und erneutes Verbinden der Python-App

Wenn die Python-App ausgeführt wird, signalisiert eine Anzeige in der Statusleiste, ob Python einsatzbereit ist. Bis die Verbindung hergestellt ist, reagiert die CE-Tastatur möglicherweise nicht. Es empfiehlt sich, während Ihrer Python-Sitzung auf die Verbindungsanzeige in der Statusleiste zu achten.



Python nicht bereit

Python bereit

Bildschirmaufnahmen

Unter Verwendung von TI Connect™ CE unter education.ti.com/84cetupdate sind Bildschirmaufnahmen von jedem Python-App-Bildschirm erlaubt.

Python-Arbeitsbereiche

Die Python-App enthält drei Arbeitsbereiche für Ihre Python-Programmentwicklung.

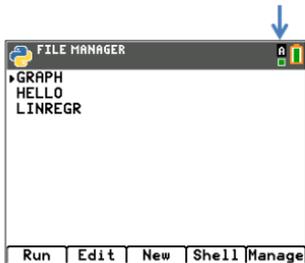
- [Dateimanager](#)
- [Editor](#)
- [Shell](#)

Python-Dateimanager

Der Dateimanager listet die im RAM Ihres Taschenrechners verfügbaren Python AppVars auf. Sie können Programme erstellen, bearbeiten und ausführen sowie zur Shell navigieren.

Drücken Sie im Alpha-Status einen beliebigen Buchstaben auf der Tastatur, um zu Programmen zu springen, die mit diesem Buchstaben beginnen.

Drücken Sie bei Bedarf **[alpha]**, wenn die Anzeige **A** nicht in der Statusleiste angezeigt wird.



Tastenkürzel und Menüs im Dateimanager

Menüs	Taste	Beschreibung
[Run]	[y=]	Wählen Sie mit [↑] oder [↓] ein Programm aus. Wählen Sie dann [Run], um das Programm auszuführen.
[Edit]	[window]	Wählen Sie mit [↑] oder [↓] ein Programm aus. Wählen Sie dann [Edit], um das Programm im Editor anzuzeigen und zu bearbeiten.
[New]	[zoom]	Wählen Sie [New], um einen neuen Programmnamen einzugeben und zum Editor zu wechseln, um Ihr neues Programm einzugeben. Wählen Sie im Bildschirm [New] die Option [Types] (drücken Sie hierzu [zoom]), um einen Programmtyp auszuwählen. Durch die Auswahl eines Programmtyps wird eine Vorlage von Import-Anweisungen und häufig verwendeten Funktionen und Methoden für diese Aktivität in Ihr neues Programm eingefügt.
[Shell]	[trace]	Wählen Sie [Shell], um die Shell-Eingabeaufforderung anzuzeigen (Python-Interpreter). Die Shell wird im aktuellen Zustand angezeigt.
[Manage]	[graph]	Wählen Sie [Manage], um: <ul style="list-style-type: none">• die Versionsnummer anzuzeigen.

Tastenkürzel und Menüs im Dateimanager

Menüs	Taste	Beschreibung
		<ul style="list-style-type: none">• ein ausgewähltes Programm zu kopieren, zu löschen oder umzubenennen.• den Info-Bildschirm anzuzeigen.• die App zu beenden. Verwenden Sie hierzu auch [2nd] [quit]

Erstellen eines neuen Programms mit den Programmtyp-Vorlagen

Selection pastes appropriate import statement(s) and program lines to the new program.

- Select [Types] to display Select Program Type menu.
- Import(s) displays on status bar.

Erstellen eines neuen STEM Activity-Programms mit Vorlagen

Wenn die AppVar TISTEMEN im Archiv geladen ist, wird der Menüpunkt „TI STEM Project Helpers...“ im Menü „Select Program Type“ aufgeführt. Wählen Sie die gewünschte STEM Activity-Vorlage, die Sie bei der Erstellung des neuen MINT-Programms unterstützt.



Python-Editor

Der Python-Editor wird in einem ausgewählten Programm im Dateimanager oder in der Shell angezeigt. Der Editor zeigt Schlüsselwörter, Operatoren, Kommentare, Strings und Einzüge in Farbe an. Schnelles Einfügen gängiger Python-Schlüsselwörter und -Funktionen ist ebenso möglich wie die direkte Tastatureingabe und die Eingabe von `[a A #]`-Zeichen. Beim Einfügen eines Codeblocks wie z. B. `if.. else` bietet der Editor eine automatische Einrückung, die beim Schreiben des Programms nach Bedarf geändert werden kann.

Der Cursor befindet sich immer im Einfügemodus. Verwenden Sie `[2nd]` und `[alpha]`, um den Cursor umzuschalten. Die Cursor-Status sind numerisch, `a` und `A`. `[del]` funktioniert wie eine Rücktaste und löscht ein Zeichen.

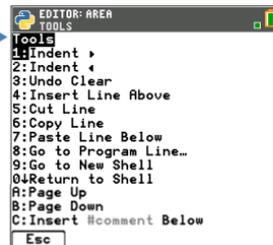
Position des Cursors in der Programmzeile.

Automatische Einrückung bei Codeblocks.

Graue Punkte als visueller Indikator für eingerückte Zeilen.



Nützliche Werkzeuge für die Bearbeitung und das Arbeiten in der Shell. Vollständige Beschreibung folgt.



Tastenkürzel und Menüs im Python-Editor

Menüs	Taste	Beschreibung
[Fns...]	<code>[y=]</code>	Wählen Sie [Fns...], um auf Menüs mit häufig verwendeten Funktionen, Schlüsselwörtern und Operatoren zuzugreifen. Sie können so auch auf ausgewählte Inhalte der Mathematik- und

Tastenkürzel und Menüs im Python-Editor

Menüs	Taste	Beschreibung																
		Zufallsmodule zugreifen. Hinweis: <code>[2nd]</code> <code>[catalog]</code> ist auch für das schnelle Einfügen hilfreich.																
[a A #]	<code>[window]</code>	Wählen Sie <code>[a A #]</code> , um auf eine Zeichenpalette als alternative Möglichkeit zur Eingabe vieler Zeichen zuzugreifen.																
[Tools]	<code>[zoom]</code>	Wählen Sie [Tools], um auf Funktionen zuzugreifen, die Sie bei der Bearbeitung oder Interaktion mit der Shell unterstützen. <table border="1" style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="width: 20%;">1: Indent ▶</td> <td>Rückt die Programmzeile nach rechts ein, der Cursor wird auf das erste Zeichen der Zeile gesetzt.</td> </tr> <tr> <td>2: Indent ◀</td> <td>Verkleinert die Einrückung der Programmzeile nach links. Der Cursor wird auf das erste Zeichen der Zeile gesetzt.</td> </tr> <tr> <td>3: Undo Clear</td> <td>Fügt die letzte gelöschte Zeile in eine neue Zeile unterhalb der Programmzeile ein, die den Cursor enthält. Der Cursor wird am Ende der eingefügten Zeile angezeigt.</td> </tr> <tr> <td>4: Insert Line Above</td> <td>Fügt eine Zeile oberhalb der Programmzeile ein, in der sich der Cursor befindet. Ggf. wird die Zeile eingerückt und der Einzug durch Punkte angezeigt.</td> </tr> <tr> <td>5: Cut Line</td> <td>Die aktuelle Programmzeile, in der sich der Cursor befindet, wird ausgeschnitten. Der Cursor wird auf der Programmzeile angezeigt, die sich unter der ausgeschnittenen Zeile befindet.</td> </tr> <tr> <td>6: Copy Line</td> <td>Kopiert die aktuelle Programmzeile, in der sich der Cursor befindet. Eine kopierte Programmzeile kann an der Shell-Eingabeaufforderung eingefügt werden. Siehe nachstehenden Abschnitt „Shell“.</td> </tr> <tr> <td>7: Paste Line Below</td> <td>Fügt die letzte gespeicherte Programmzeile in die Zeile unter der Cursorposition ein.</td> </tr> <tr> <td>8: Go to</td> <td>Zeigt den Cursor am Anfang der</td> </tr> </tbody> </table>	1: Indent ▶	Rückt die Programmzeile nach rechts ein, der Cursor wird auf das erste Zeichen der Zeile gesetzt.	2: Indent ◀	Verkleinert die Einrückung der Programmzeile nach links. Der Cursor wird auf das erste Zeichen der Zeile gesetzt.	3: Undo Clear	Fügt die letzte gelöschte Zeile in eine neue Zeile unterhalb der Programmzeile ein, die den Cursor enthält. Der Cursor wird am Ende der eingefügten Zeile angezeigt.	4: Insert Line Above	Fügt eine Zeile oberhalb der Programmzeile ein, in der sich der Cursor befindet. Ggf. wird die Zeile eingerückt und der Einzug durch Punkte angezeigt.	5: Cut Line	Die aktuelle Programmzeile, in der sich der Cursor befindet, wird ausgeschnitten. Der Cursor wird auf der Programmzeile angezeigt, die sich unter der ausgeschnittenen Zeile befindet.	6: Copy Line	Kopiert die aktuelle Programmzeile, in der sich der Cursor befindet. Eine kopierte Programmzeile kann an der Shell-Eingabeaufforderung eingefügt werden. Siehe nachstehenden Abschnitt „Shell“.	7: Paste Line Below	Fügt die letzte gespeicherte Programmzeile in die Zeile unter der Cursorposition ein.	8: Go to	Zeigt den Cursor am Anfang der
1: Indent ▶	Rückt die Programmzeile nach rechts ein, der Cursor wird auf das erste Zeichen der Zeile gesetzt.																	
2: Indent ◀	Verkleinert die Einrückung der Programmzeile nach links. Der Cursor wird auf das erste Zeichen der Zeile gesetzt.																	
3: Undo Clear	Fügt die letzte gelöschte Zeile in eine neue Zeile unterhalb der Programmzeile ein, die den Cursor enthält. Der Cursor wird am Ende der eingefügten Zeile angezeigt.																	
4: Insert Line Above	Fügt eine Zeile oberhalb der Programmzeile ein, in der sich der Cursor befindet. Ggf. wird die Zeile eingerückt und der Einzug durch Punkte angezeigt.																	
5: Cut Line	Die aktuelle Programmzeile, in der sich der Cursor befindet, wird ausgeschnitten. Der Cursor wird auf der Programmzeile angezeigt, die sich unter der ausgeschnittenen Zeile befindet.																	
6: Copy Line	Kopiert die aktuelle Programmzeile, in der sich der Cursor befindet. Eine kopierte Programmzeile kann an der Shell-Eingabeaufforderung eingefügt werden. Siehe nachstehenden Abschnitt „Shell“.																	
7: Paste Line Below	Fügt die letzte gespeicherte Programmzeile in die Zeile unter der Cursorposition ein.																	
8: Go to	Zeigt den Cursor am Anfang der																	

Tastenkürzel und Menüs im Python-Editor

Menüs	Taste	Beschreibung												
		<table border="1"><tr><td>Program Line...</td><td>angegebenen Programmzeile an.</td></tr><tr><td>9: Go to New Shell</td><td>Zeigt die neu initialisierte Shell an.</td></tr><tr><td>0: Return to Shell</td><td>Zeigt die Shell im aktuellen Zustand an.</td></tr><tr><td>A: Page up</td><td>Zeigt 11 Programmzeilen über der aktuellen Cursorposition an, sofern verfügbar.</td></tr><tr><td>B: Page Down</td><td>Zeigt 11 Programmzeilen unter der aktuellen Cursorposition an, sofern verfügbar.</td></tr><tr><td>C: Insert #comment Below</td><td>Fügt # in eine neue Zeile unterhalb der Cursorposition ein.</td></tr></table>	Program Line...	angegebenen Programmzeile an.	9: Go to New Shell	Zeigt die neu initialisierte Shell an.	0: Return to Shell	Zeigt die Shell im aktuellen Zustand an.	A: Page up	Zeigt 11 Programmzeilen über der aktuellen Cursorposition an, sofern verfügbar.	B: Page Down	Zeigt 11 Programmzeilen unter der aktuellen Cursorposition an, sofern verfügbar.	C: Insert #comment Below	Fügt # in eine neue Zeile unterhalb der Cursorposition ein.
Program Line...	angegebenen Programmzeile an.													
9: Go to New Shell	Zeigt die neu initialisierte Shell an.													
0: Return to Shell	Zeigt die Shell im aktuellen Zustand an.													
A: Page up	Zeigt 11 Programmzeilen über der aktuellen Cursorposition an, sofern verfügbar.													
B: Page Down	Zeigt 11 Programmzeilen unter der aktuellen Cursorposition an, sofern verfügbar.													
C: Insert #comment Below	Fügt # in eine neue Zeile unterhalb der Cursorposition ein.													
[Run]	<code>trace</code>	Wählen Sie [Run], um Ihr Programm auszuführen.												
[Files]	<code>graph</code>	Wählen Sie [Files], um den Dateimanager anzuzeigen.												

Python-Shell

Die Python-Shell ist die Konsole, auf der Sie mit dem Python-Interpreter interagieren oder Ihre Python-Programme ausführen können. Schnelles Einfügen gängiger Python-Schlüsselwörter und -Funktionen ist ebenso möglich wie die direkte Tastatureingabe und die Eingabe von **[a A #]**-Zeichen. Die Shell-Eingabeaufforderung kann zum Testen einer aus dem Editor eingefügten Codezeile verwendet werden. Es können auch mehrere Codezeilen eingegeben und an einer Shell-Eingabeaufforderung `>>>` ausgeführt werden.

Cursorstatus-Indikator der Shell.

Die Shell wird reinitialisiert, wenn ein neues Programm ausgeführt wird.



Nützliche Werkzeuge für das Arbeiten in der Shell. Siehe nachstehende Informationen.



Shell-Cursorstatus

nicht alphabetisch
[2nd alpha],
wenn ein Umschalten erforderlich ist

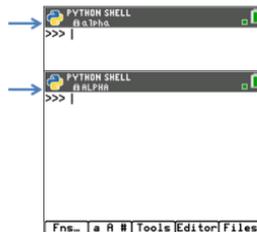


[alpha]
alphabetisch

[alpha]
erneut
ALPHA

[2nd alpha]
alphabetisch feststellen

erneut
[alpha]
ALPHA feststellen



Tastenkürzel und Menüs in der Python-Shell

Menüs	Taste	Beschreibung																		
[Fns...]	y=	Wählen Sie [Fns...], um auf Menüs mit häufig verwendeten Funktionen, Schlüsselwörtern und Operatoren zuzugreifen. Sie können so auch auf ausgewählte Inhalte der Mathematik- und Zufallsmodule zugreifen. Hinweis: 2nd [catalog] ist auch für das schnelle Einfügen hilfreich.																		
[a A #]	window	Wählen Sie [a A #] , um auf eine Zeichenpalette als alternative Möglichkeit zur Eingabe vieler Zeichen zuzugreifen.																		
[Tools]	zoom	Wählen Sie [Tools], um die folgenden Menüpunkte anzuzeigen: <table border="1" data-bbox="453 498 940 1329"> <tbody> <tr> <td>1: Rerun last program</td> <td>Führt das letzte Programm, das in der Shell ausgeführt wurde, erneut aus.</td> </tr> <tr> <td>2: Run...</td> <td>Zeigt eine Liste der zur Ausführung in der Shell verfügbaren Python-Programme an.</td> </tr> <tr> <td>3: Paste from Editor</td> <td>Fügt die letzte kopierte Programmzeile aus dem Editor in die Shell-Eingabeaufforderung ein.</td> </tr> <tr> <td>4: Vars...</td> <td>Zeigt die Variablen des zuletzt ausgeführten Programms an. Zeigt keine programmdefinierten Variablen aus einem importierten Programm an.</td> </tr> <tr> <td>5: Clear Screen</td> <td>Löscht den Shell-Bildschirm. Eine neue Shell wird nicht reinitialisiert.</td> </tr> <tr> <td>6: New Shell</td> <td>Reinitialisiert eine neue Shell.</td> </tr> <tr> <td>7: Go to Program Line...</td> <td>Zeigt den Editor von der Shell aus mit dem Cursor auf der angegebenen Programmzeile an.</td> </tr> <tr> <td>8: Last Entry>>> 2nd ↓</td> <td>Zeigt während einer Shell-Sitzung die letzten Einträge an der Shell-Eingabeaufforderung an (bis zu 8).</td> </tr> <tr> <td>9: View History 2nd ↑ 2nd ↓</td> <td>Führen Sie auf dem Shell-Bildschirm einen Bildlauf durch, um die letzten während einer Shell-Sitzung ausgegebenen Zeilen anzuzeigen (bis zu 60). Nachdem Sie mit ti_plotlib, ti_draw oder ti_image in der Shell</td> </tr> </tbody> </table>	1: Rerun last program	Führt das letzte Programm, das in der Shell ausgeführt wurde, erneut aus.	2: Run...	Zeigt eine Liste der zur Ausführung in der Shell verfügbaren Python-Programme an.	3: Paste from Editor	Fügt die letzte kopierte Programmzeile aus dem Editor in die Shell-Eingabeaufforderung ein.	4: Vars...	Zeigt die Variablen des zuletzt ausgeführten Programms an. Zeigt keine programmdefinierten Variablen aus einem importierten Programm an.	5: Clear Screen	Löscht den Shell-Bildschirm. Eine neue Shell wird nicht reinitialisiert.	6: New Shell	Reinitialisiert eine neue Shell.	7: Go to Program Line...	Zeigt den Editor von der Shell aus mit dem Cursor auf der angegebenen Programmzeile an.	8: Last Entry>>> 2nd ↓	Zeigt während einer Shell-Sitzung die letzten Einträge an der Shell-Eingabeaufforderung an (bis zu 8).	9: View History 2nd ↑ 2nd ↓	Führen Sie auf dem Shell-Bildschirm einen Bildlauf durch, um die letzten während einer Shell-Sitzung ausgegebenen Zeilen anzuzeigen (bis zu 60). Nachdem Sie mit ti_plotlib, ti_draw oder ti_image in der Shell
1: Rerun last program	Führt das letzte Programm, das in der Shell ausgeführt wurde, erneut aus.																			
2: Run...	Zeigt eine Liste der zur Ausführung in der Shell verfügbaren Python-Programme an.																			
3: Paste from Editor	Fügt die letzte kopierte Programmzeile aus dem Editor in die Shell-Eingabeaufforderung ein.																			
4: Vars...	Zeigt die Variablen des zuletzt ausgeführten Programms an. Zeigt keine programmdefinierten Variablen aus einem importierten Programm an.																			
5: Clear Screen	Löscht den Shell-Bildschirm. Eine neue Shell wird nicht reinitialisiert.																			
6: New Shell	Reinitialisiert eine neue Shell.																			
7: Go to Program Line...	Zeigt den Editor von der Shell aus mit dem Cursor auf der angegebenen Programmzeile an.																			
8: Last Entry>>> 2nd ↓	Zeigt während einer Shell-Sitzung die letzten Einträge an der Shell-Eingabeaufforderung an (bis zu 8).																			
9: View History 2nd ↑ 2nd ↓	Führen Sie auf dem Shell-Bildschirm einen Bildlauf durch, um die letzten während einer Shell-Sitzung ausgegebenen Zeilen anzuzeigen (bis zu 60). Nachdem Sie mit ti_plotlib, ti_draw oder ti_image in der Shell																			

Tastenkürzel und Menüs in der Python-Shell

Menüs	Taste	Beschreibung
		<p>gezeichnet haben, können Sie mit [clear] die Zeichnung wieder löschen und zur Shell zurückkehren. Das Protokoll wird nicht angezeigt. Verwenden Sie die Tasten [2nd]  und [2nd] , um das Protokoll einzusehen.</p> <hr/> <p>0: Tab Complete  [enter]</p> <p>Zeigt die Namen der Variablen und Funktionen an, auf die in der aktuellen Shell-Sitzung zugegriffen werden kann.</p> <p>Wenn ein Buchstabe einer verfügbaren Variable oder Funktion eingegeben wird, drücken Sie  [enter], um den Namen automatisch auszufüllen, sofern in der aktuellen Shell-Sitzung eine Übereinstimmung verfügbar ist.</p> <hr/> <p>A: from PROGRAM import *...</p> <p>Bei der erstmaligen Ausführung in einer Shell-Sitzung wird PROGRAM ausgeführt, und Variablen sind nur unter Verwendung von „Tab Complete“ sichtbar.</p> <p>Wenn das Programm in derselben Shell-Sitzung erneut ausgeführt wird, wird die Ausführung nicht ausgeführt.</p> <p>Dieser Befehl kann auch über  [catalog] eingefügt werden.</p>
[Editor]		Wählen Sie [Editor], um den Editor mit den letzten Programmen im Editor anzuzeigen. Wenn der Editor leer ist, können Sie den Dateimanager anzeigen.
[Files]		Wählen Sie [Files], um den Dateimanager anzuzeigen.

Hinweis:

- Um ein laufendes Python-Programm zu unterbrechen, z. B. wenn sich ein Programm in einer Endlosschleife befindet, drücken Sie . Alternativ können Sie **[Tools] () > 6:New Shell** drücken, um ein laufendes Programm anzuhalten.
- Wenn Sie mit den Modulen `ti_plotlib`, `ti_draw` oder `ti_image` in der Shell (Konsole) zeichnen, können Sie über die Taste [clear] die Zeichnung wieder löschen und zur Shell-Eingabeaufforderung oben auf dem Bildschirm zurückkehren. Verwenden Sie die Tasten [2nd]  und [2nd] , um das Shell-Protokoll einzusehen.

Ausführungsfehler: mit Shell >Tools zur Programmzeile gehen

Die TI-Python-Umgebung zeigt Python-Fehlermeldungen in der Shell an, wenn der Code ausgeführt wird. Wenn bei der Ausführung eines Programms ein Fehler angezeigt wird, wird eine Programmzeilennummer angezeigt. Verwenden Sie **Shell>Tools 7:Go to Program Line...** Geben Sie die Zeilennummer ein und drücken Sie **[OK]**. Der Cursor wird im Editor auf dem ersten Zeichen der entsprechenden Programmzeile angezeigt. Die Programmzeilennummer wird in der zweiten Zeile der Statusleiste im Editor angezeigt.

Unterstützung der Schnellbearbeitung

Wenn Sie Code im Editor oder in der Shell eingeben, verwenden Sie die folgenden Eingabemethoden, um diesen schnell in die Bearbeitungszeile einzufügen.

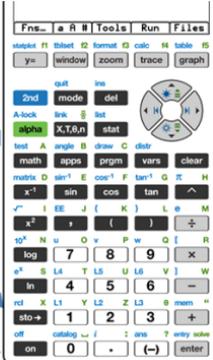
Tipps für die schnelle Eingabe

- [Verwenden der Python-Tastatur](#)
- [Verwenden des Python-Katalogs](#)
- [Verwenden der \[a A #\] Zeichentabelle](#)

Verwenden der Python-Tastatur

Wenn die Python-Anwendung läuft, ist die Tastatur so gestaltet, dass die entsprechenden Python-Operationen eingefügt oder Menüs geöffnet werden können, die für die einfache Eingabe von Funktionen, Schlüsselwörtern, Methoden, Operatoren usw. konzipiert sind. Wenn Sie **[2nd]** und **[alpha]** drücken, erhalten Sie wie im Betriebssystem Zugriff auf die zweite und dritte Funktion einer Taste.

Navigation, Bearbeiten und Sonderzeichen in der Python-App nach Tastaturzeilen

Navigation in der App		Funktasten <ul style="list-style-type: none">• Navigation in der Editor-Zeile.• Navigation in der Shell-Eingabeaufforderung und im Verlaufs.• Bildschirm-Helligkeit.• [2nd] [-<] oder [->] zum Anfang oder Ende der Zeile.
[2nd] Zugriff auf Zweitfunktion der Taste. [2nd] [quit] App beenden. [del] Rücktaste in der Eingabezeile. [del] Im Dateimanager löschen.		[clear] löscht eine Eingabezeile oder den Bildschirm „Info“. [clear] löscht keine Menüs. ([Esc] in der App.) [clear] löscht in der Shell einen Plot bzw. ein Diagramm
[alpha] schaltet den Cursor-Status um: nicht alphabetisch, alpha und ALPHA [2nd] [alpha] sperrt einen Alpha-Status. Wählen Sie Buchstaben über die Tastatur aus.		Klammern und Satzzeichen <ul style="list-style-type: none">[2nd] [[]] oder [[]][2nd] [[]] oder [[]][,]
[2nd] [link] fügt ein „\“ ein		[2nd] [L3] fügt ein „#“ ein [alpha] [0] fügt ein „@“ ein [alpha] ["] fügt ein doppeltes Anführungszeichen ein [2nd] [mem] fügt ein einzelnes Anführungszeichen ein
[2nd] [off] schaltet den CE aus. Die App wird geschlossen. Die Python-Sitzung wird als neue Sitzung reinitialisiert, wenn die App erneut gestartet wird. [on] schaltet den CE ein; schaltet Auto-Dim aus; schaltet CE aus APD* wieder ein. Die Python-Sitzung wird nach Auto-Dim und APD wiederaufgenommen. [on] bricht ein Programm ab, wenn es in der Shell läuft.		[alpha] [space] fügt ein Leerzeichen ein [.] fügt einen Punkt oder Dezimalpunkt ein [2nd] [ans] fügt einen Unterstrich („_“) ein [alpha] [?] fügt ein Fragezeichen ein [2nd] [entry] Tab Complete (Registerkarte vollständig) (Shell-Tools)

Spezifische Tastenbetätigungen für Menüs und Funktionen in der Python-App nach Tastaturzeilen

The image shows a screenshot of the Python-App interface with several callouts explaining keyboard shortcuts for various menu items and functions:

- [X,T,θ,n] X oder x [2nde] [listes] Menü „Liste“
- [math] Menü „Module“ [2nd] [test] Menü „Operatoren“
- [xⁿ-1] ******-1 wird eingefügt
- [2nd] [rc] zeigt das Menü `ti_system` an Import des Moduls `ti_system`
- [2nd] [catalog] zeigt den Python-spezifischen Katalog an.
- [2nd] [i] zeigt Complex Type Imaginary (imaginärteile komplexer Zahlen) an (`ja+bi`).
- [var] zeigt verfügbare Variablen in Shell an, nachdem ein Programm ausgeführt wurde.
- [2nd] [π] [sin]; [cos]; [tan]; [2nd][sin]; [2nd][cos]; [2nd][tan] zeigt das Menü „Trig“ an; Math-Modul importieren

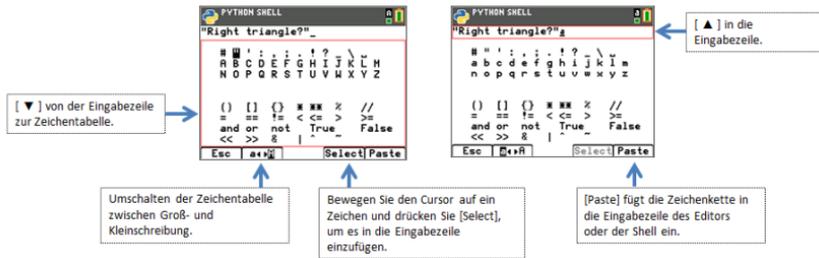
Spezifische Tastenbetätigungen für Menüs und Funktionen in der Python-App nach Tastaturzeilen (Forts.)

The image shows a screenshot of the Python-App interface with several callouts explaining keyboard shortcuts for mathematical operations and menu actions:

- [xⁿ2] fügt ****2** ein [2nd] [√] fügt `sqrt()` ein [2nd] [EE] fügt `E` ein
- [log] fügt `log(,10)` ein [2nd] [10^x] fügt `10**` ein
- [ln] fügt `log()`Basis `e` ein [2nd] [e^x] fügt `exp()` ein
- [sto >] fügt ein „`=`“ ein
- [var] zeigt verfügbare Variablen in Shell an, nachdem ein Programm ausgeführt wurde. [clear] Löscht den Bereich der grafischen Darstellung in Shell für `ti_plotlib` Plotting-Methoden.
- [^] fügt ****** ein
- [→] fügt ein „`r`“ ein [2nde] [e] fügt ein „`e`“ ein
- [*] fügt ein ***** ein
- [.] fügt ein „`e`“ ein
- [+] fügt ein „`+`“ ein
- [enter]
 - Führt im File Manager (Dateimanager) das ausgewählte Programm aus.
 - Teilt im Editor eine Programmzeile.
 - Verwenden Sie [2nd] [enter] , um eine Zeile darunter einzufügen.

Verwenden der [a A #] Zeichentabelle

Die [a A #]-Verknüpfungs-Registerkarte zu einer Zeichenpalette ist eine praktische Funktion zur Eingabe von Zeichenketten im Editor oder in der Shell.



Hinweis: Wenn sich der Cursor-Fokus in der [a A #]-Eingabezeile befindet, sind ausgewählte Tasten der Tastatur nicht verfügbar. Wenn sich der Fokus in der Zeichentabelle befindet, ist das Tastenfeld eingeschränkt.

[Fns...] Menüs, Module und Add-on-Module

- [\[Fns...\] Menüs](#)
- [\[Fns...\] Integriert, Operatoren und Schlüsselwörter](#)
- [\[Fns...\] Module](#)
- [\[Fns...\] Add-on-Module](#)

[Fns...] Menüs

Die Verknüpfungs-Registerkarte [Fns...] zeigt Menüs mit häufig verwendeten Python-Funktionen, Schlüsselwörtern und Operatoren an. Die Menüs bieten auch Zugang zu den ausgewählten Funktionen und Konstanten aus den Modulen und Add-on-Modulen. Anstatt dass Sie Zeichen für Zeichen über die Tastatur eingeben, bieten diese Menüs eine schnelle Möglichkeit zum Einfügen in Editor oder Shell. Drücken Sie [Fns...], wenn Sie sich im Editor oder in der Shell befinden. Alternative Eingabemethoden finden Sie auch unter Verwenden des Python-Katalogs und Verwenden der Python-Tastatur.

[Fns...] Integriert, Operatoren und Schlüsselwörter

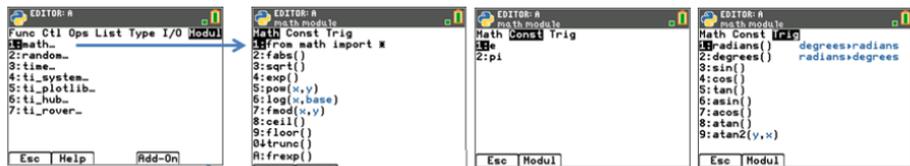


[Fns...] Module

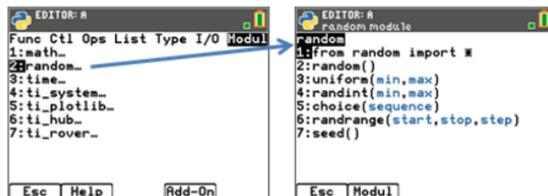
Wenn Sie eine Python-Funktion oder -Konstante aus einem Modul verwenden, verwenden Sie immer eine Import-Anweisung, um den Ort der Funktion, Methode oder Konstante im Modul anzugeben.

Siehe [Was ist die Python-Programmierung?](#)

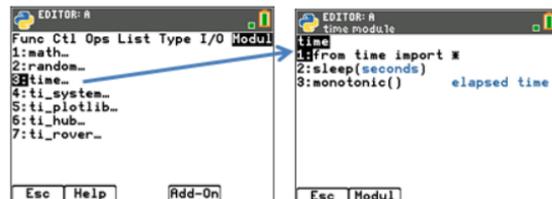
[Fns...]>Modul: math



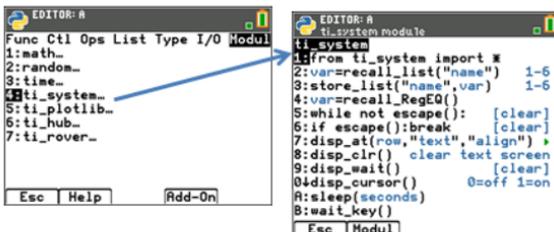
[Fns...]>Modul: random



[Fns...]>Modul: time



[Fns...]>Modul: ti_system



Siehe: Tastaturbelegung für wait_key()

[Fns...]>Modul: ti_plotlib



Wichtiger Hinweis für die grafische Darstellung:

- Um die erwarteten Ergebnisse zu gewährleisten, muss die Reihenfolge der Programmzeilen für das Plotting der Reihenfolge im Setup-Menü folgen.
- Die grafische Anzeige wird angezeigt, wenn `plt.show_plot()` am Ende der Plotting-Objekte in einem Programm ausgeführt wird. Um den Plotting-Bereich in der Shell zu löschen, drücken Sie `[clear]`. Verwenden Sie die Tasten `[2nd]` `[↑]` und `[2nd]` `[↓]`, um das Shell-Protokoll einzusehen.
- Die Ausführung eines zweiten Programms, das davon ausgeht, dass die Standardwerte innerhalb derselben Shell-Umgebung festgelegt sind, führt im Allgemeinen zu unerwartetem Verhalten, beispielsweise bei den Farb- oder anderen Standardargument-Einstellungen. Bearbeiten Sie Programme mit erwarteten Argumentwerten oder reinitialisieren Sie die Shell, bevor Sie ein anderes Plotting-Programm ausführen.

[Fns...]>Modul: ti_hub module

`ti_hub`-Methoden sind nicht im Katalog und somit auch nicht im Referenz-Leitfaden aufgeführt. Bitte benutzen Sie für Argumente und Argumentvorgaben oder Details zu erlaubten Werten die Bildschirminformationen in den Menüs. Weitere Informationen zur Python-Programmierung für TI-Innovator™ Hub und TI-Innovator™ Rover stehen unter education.ti.com zur Verfügung.

Hinweis: TI-Innovator™ Hub sollte angeschlossen sein, wenn Sie Ihre Python-Programme ausführen.

```

EDITOR: #
Hub Modules
Func Ctl Ops List Type I/O Modul
1: math..
2: random..
3: time..
4: ti_system..
5: ti_plotlib..
6: ti_hub..
7: ti_rover..
Esc Help Add-On

```

```

EDITOR: #
Hub Modules
Import Commands Ports Advanced
1: Hub Built-in devices..
2: Input devices..
3: Output devices..
4: Collect data..
Esc Modul

```

```

EDITOR: #
Import Commands Ports Advanced
1: from ti_system import *
2: sleep(seconds)
3: disp_at(row,"text","align")
4: disp_clr() clear text screen
5: disp_wait() [clear]
6: disp_cursor() Onoff ison
7: while not escape(): [clear]
Esc Modul

```

```

EDITOR: #
Import Commands Ports Advanced
1: OUT 1
2: OUT 2
3: OUT 3
4: IN 1
5: IN 2
6: IN 3
7: BB 1
8: BB 2
9: BB 3
0: BB 4
Esc Modul

```

```

EDITOR: #
Import Commands Ports Advanced
1: from ti_hub import *
2: connect("obj","arg")
3: disconnect("obj","arg")
4: set("obj","arg")
5: read("obj","arg")
6: calibrate("obj","arg")
7: range("obj","arg")
8: version()
9: begin()
0: start()
Esc Modul

```

```

EDITOR: #
Paits = Color > Moduls menu
Hub Built-in devices
1: Color RGB LED Output
2: Light Red LED Output
3: Sound Sound Output
4: Brightness Light Sensor Input
Esc Import

```

```

EDITOR: #
Paits = DHT > Moduls menu
Input devices
1: DHT Digital Humidity & Temp
2: Ranger
3: Light Level
4: Temperature
5: Moisture
6: Magnetic
7: Vernier TI-SensorLink Input
8: Analog in
9: Digital in
0: Potentiometer
R: Thermistor
B: Loudness
C: Color Input
D: BB Port Breadboard Port
E: Hub Time Time Count from Hub
F: TI-RGB Array Input/Output
G: var.release()
Esc Import

```

```

EDITOR: #
Paits = LED > Moduls menu
Output devices
1: LED
2: RGB
3: TI-RGB Array Input/Output
4: Speaker Speaker Output
5: Power
6: Continuous Servo
7: Analog out
8: Vibration Motor
9: Relay
0: Servo
R: Squarewave
B: Digital out
C: BB Port Breadboard Port
D: var.release()
Esc Import

```

```

Collect data...
EDITOR: #
Sensors Collect Option Advanced
1: Brightness Light Sensor Input
2: DHT Digital Humidity & Temp
3: Ranger Distance (m) Light Sensor
4: Light Level Light Sensor Degrees C
5: Temperature
6: Moisture
7: Magnetic
8: Vernier TI-SensorLink Input
9: Analog in
R: Potentiometer
B: Thermistor
C: Loudness Breadboard Port
D: BB Port Breadboard Port
Esc Modul

```

```

EDITOR: #
Sensors Collect Option Advanced
1: collect()
2: set_sensors(sensors_ 1-4
3: set_time(time) t>0,<=100s
4: set_rate(rate) r>0,<=10/s
5: set_max(max) True/False
6: start()
7: varc.measurements(ansr,opt)
8: varc.measurements("ansr,opt)
9: varc.measurements("time")
Esc Modul

```

```

EDITOR: #
Sensors Collect Option Advanced
1: DHT Temperature
2: DHT Humidity
3: Magnetic Level
4: Ranger Time of Flight
Esc Modul

```

```

EDITOR: #
Sensors Collect Option Advanced
1: start(false)
2: while not c.done():
3: sleep(seconds)
Esc Modul

```

ti_hub-Modul – Import zum Editor hinzufügen und ti_hub-Sensormodul zum Modul-Menü hinzufügen

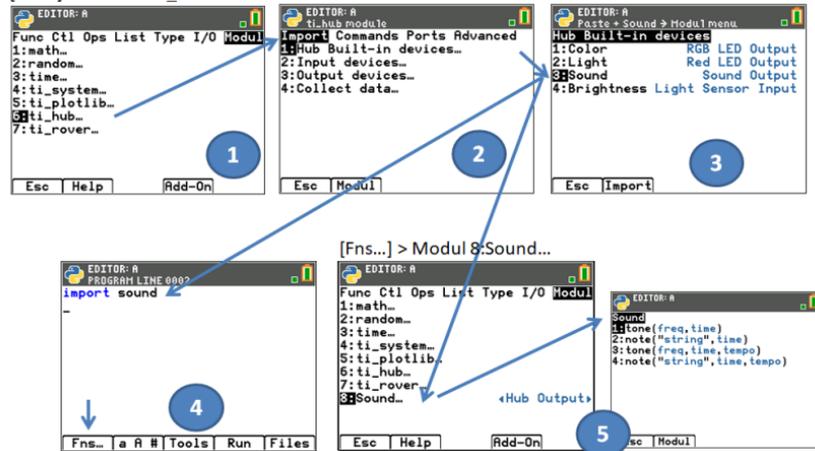
Beispiel: Ton importieren

Gehen Sie wie folgt vor, um TI-Innovator™ Sensormethoden aus dem Editor in Ihr Python-Programm zu importieren:

1. Wählen Sie **[Fns...]** > **Modul 6:ti_hub**
2. Wählen Sie das ti_hub-Importmenü. Wählen Sie einen Sensortyp aus Built-in, Input und Output.
3. Wählen Sie einen Sensor.
4. Eine Import-Anweisung wird in den Editor eingefügt und das Sensormodul steht in **[Fns...]** > **Modul** zur Verfügung, wenn Sie von Ihrem Programm zu diesem Menü zurückkehren.
5. Wählen Sie **[Fns...]** > **Modul 8:Sound...**, um geeignete Methoden für diesen Sensor einzufügen.

[Fns...]>Modul 6:ti_hub

[Fns...]> Modul 6:ti_hub



Hinweis: „Brightns“ ist ein „integriertes“ Objekt in TI-Innovator Hub.

Wenn Sie die Anweisung „import brightns“ verwenden, geben Sie „brightns.range(0,100)“ ein, um sicherzustellen, dass zu Beginn der Programmausführung der korrekten Standardbereich eingestellt ist.

Beispiel:

```
import brightns
brightns.range(0,100)
b=brightns.measurement()
print(b)
```

[Fns...]>Modul ti_rover module

ti_rover-Methoden sind nicht im Katalog und somit auch nicht im Referenz-Leitfaden aufgeführt. Bitte benutzen Sie für Argumente und Argumentvorgaben oder Details zur erlaubten Werten die Bildschirminformationen in den Menüs. Weitere Informationen zur Python-Programmierung für TI-Innovator™ Hub und TI-Innovator™ Rover stehen unter education.ti.com zur Verfügung.

The image displays a sequence of six screenshots from the TI-Innovator Hub interface, illustrating the process of navigating to and viewing the 'ti_rover' module.

- Screenshot 1:** Shows the 'Modul' menu with 'ti_rover' selected. The menu items are: 1: math..., 2: random..., 3: time..., 4: ti_system..., 5: ti_plotlib..., 6: ti_hub..., 7: ti_rover...
- Screenshot 2:** Shows the 'ti_rover module' editor. The 'Methods' list includes: Drive, I/O, Settings, Commands, 1: import ti_rover as rv, 2: forward(distance) unit, 3: backward(distance) unit, 4: left(angle) degrees, 5: right(angle) degrees, 6: stop(), 7: resume(), 8: stay(time) seconds, 9: to_xy(x,y), 10: to_polar(r,theta) degrees, 11: to_angle(angle) degrees, 12: forward_time(time) seconds, 13: backward_time(time) seconds, 14: left(angle, "unit") degrees, 15: right(angle, "unit") degrees, 16: forward_time(T,S, "unit") seconds, 17: backward_time(T,S, "unit") seconds, 18: forward(D, "unit", S, "unit") degrees, 19: backward(D, "unit", S, "unit") degrees, 20: disconnect(rv()) Disconnect.
- Screenshot 3:** Shows the 'Inputs' menu for the 'ti_rover module'. The items are: 1: ranger_measurement() meters, 2: color_measurement() 1-9, 3: red_measurement() 0-255, 4: green_measurement() 0-255, 5: blue_measurement() 0-255, 6: gray_measurement() 0-255, 7: encoders_gyro_measurement() degrees, 8: gyro_measurement() degrees, 9: ranger_time() seconds.
- Screenshot 4:** Shows the 'I/O' menu for the 'ti_rover module'. The items are: 1: units/s, 2: m/s, 3: revs/s, 4: units, 5: m, 6: revs, 7: degrees, 8: radians, 9: grads, A: counterclockwise.
- Screenshot 5:** Shows the 'Outputs' menu for the 'ti_rover module'. The items are: 1: color_rgb(r,g,b) 0-255, 2: color_blink(freq,time), 3: color_off(), 4: motor_left(speed,time) ±255, 5: motor_right(speed,time) ±255, 6: motors("ldir",L,"rdir",R,T).
- Screenshot 6:** Shows the 'Path' menu for the 'ti_rover module'. The items are: 1: waypoint_xythdrn(), 2: waypoint_prev(), 3: waypoint_eta(), 4: path_done(), 5: pathlist_x(), 6: pathlist_y(), 7: pathlist_time(), 8: pathlist_heading(), 9: pathlist_distance(), 10: pathlist_revs(), 11: pathlist_cndnum(), 12: pathlist_cndnum(), 13: waypoint_x(), 14: waypoint_y(), 15: waypoint_time(), 16: waypoint_heading(), 17: waypoint_distance(), 18: waypoint_revs().

Hinweise:

- In der TI-Python-Programmierung müssen Sie keine Methoden zum Verbinden und Trennen von TI-Innovator™ Rover einschließen. Die TI-Innovator™ Rover-Python-Methoden schließen das Verbinden und Trennen ohne zusätzliche Methoden ein. Dies ist ein kleiner Unterschied zur Programmierung des TI-Innovator™ Rovers in TI-Basic.
- `rv.stop()` wird als Pause ausgeführt, anschließend werden die Rover-Bewegungen in der Warteschlange über „resume“ fortgeführt. Wenn nach `rv.stop()` ein weiterer Bewegungsbefehl ausgeführt wird, wird die Bewegungswarteschlange geleert. Auch dies ist ein kleiner Unterschied zur Programmierung des TI-Innovator™ Rovers in TI-Basic.

Python-Support für TI-Innovator Sketch v1.5

 ti_hub module	Built-in... > Sound Module EDITOR: 0 Sound 1: tone(freq, time) 2: note("string", time) 3: tone(freq, time, tempo) 4: note("string", time, tempo) Esc Modul	Output > Speaker Module EDITOR: 0 Speaker 1: var::speaker("port") > 2: var::tone(freq, time) 3: var::note("string", time) 4: var::tone(freq, time, tempo) 5: var::note("string", time, tempo) Esc Modul	Input/Output > TI-RGB Module EDITOR: 0 TI-RGB Array 1: var::rgb_array() 2: var::set_led_position(r,g,b) 3: var::set_all(r,g,b) 0-255 4: var::all_off() 5: var::pattern(val) 0-65535 6: var::measurement() mA current 7: var::set_led_list(r,g,b) 8: var::pattern(val,r,g,b) 0-255 Esc Modul
 ti_hub module	Input > Ranger Module EDITOR: 0 Ranger 1: var::ranger("port") > 2: var::measurement_time() Esc Modul	Input > Magnetic Module (fix) EDITOR: 0 Magnetic 1: var::magnetic("port") > 2: var::magnet_time() 3: var::measurement() 4: var::trigger(val) 0-16383 Esc Modul	ti_rover > I/O > Inputs EDITOR: 0 ti_rover module Inputs 1: ranger_measurement() meters 2: color_measurement() 1-9 3: red_measurement() 0-255 4: green_measurement() 0-255 5: blue_measurement() 0-255 6: gray_measurement() 0-255 7: encoders_gyro_measurement() 8: gyro_measurement() degrees 9: ranger_time() seconds Esc I/O 

[Fns...] Add-on-Module

Add-on-Module verbessern das Benutzererlebnis der Python-App durch zusätzliche Funktionen und bieten einfachen Zugriff auf weitere Python-Methoden über die Menüs der Python-App.

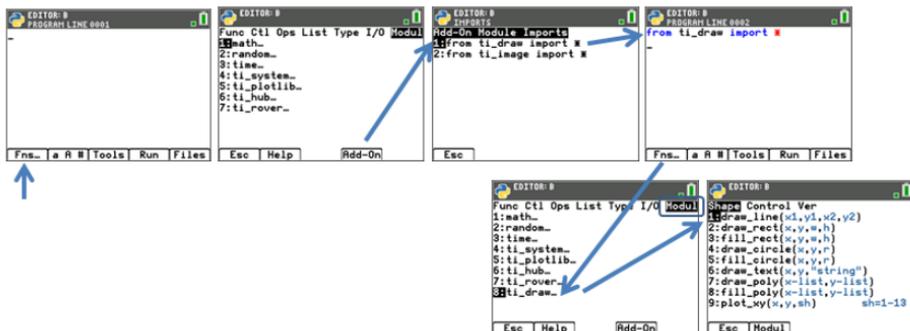
Je nach Region stehen über TI Connect™ CE ggf. Add-on-Module wie `ce_turtl`, `ce_chart`, `ce_box`, `ce_quivr` und `microbit` im Rahmen von Python-Materialien auf education.ti.com zum Download zur Verfügung. Sie müssen über die neueste Version der kürzlich geposteten Add-on-Module verfügen. Einige Add-on-Module wie etwa `ti_draw` und `ti_image` laden sich mit dem neuesten Update des CE-Bundles auf Ihren Taschenrechner.

Die Python-App zeigt das Menü der Add-on-Module im Menü [Fns...] > Modul nur dann an, wenn Ihr Programm im Editor mit einer entsprechenden Importanweisung startet.

Einfügen einer Add-on-Modul-Importanweisung in den Editor

Schritte:

1. Erstellen Sie ein neues Programm.
2. Wählen Sie im Editor [Fns...] > Modul aus.
3. Wählen Sie [Add-On] aus. Wenn ein Add-on-Modul auf den Taschenrechner geladen wurde, wird ein Importanweisungsmenü für heruntergeladene Module angezeigt.
4. Wählen Sie die Importanweisung aus, die Sie in den Editor einfügen möchten.
5. Wählen Sie [Fns...] > Modul, um zu den Menüs des über „Add-On“ importierten Add-on-Moduls zu gelangen.



Hinweise:

- [Add-On Modules Imports...] wird auch unter [2nd][catalog] aufgeführt.
- Add-on-Module sind im Archiv gespeicherte „AppVar“-Taschenrechnerdateien, die unter [mem] als eine AppVar erscheinen. Für ein optimales Benutzererlebnis der Python-App-Module empfehlen wir, diese Dateien im Archivspeicher zu lassen.
- Ein Python-Programm lässt sich dann in der Python-App über den Dateimanager oder Editor ausführen, wenn sich das „PY-AppVar“-Programm im RAM befindet. Wenn ein PY-AppVar-Python-Programm im Archivspeicher ist, kann es in der Python-App nicht ausgeführt oder bearbeitet werden.

[Fns...] ti_draw Add-on-Modul

Das ti_draw-Modul ist im neuesten CE-Bundle enthalten. Fügen Sie die Importanweisung über [Fns...] > Modul [Add-On] in Ihr Programm ein. Das ti_draw-Menü wird dann im Menü von [Fns...] > Modul folgendermaßen angezeigt.

```

EDITOR: D
PROGRAM LINE 0002
from ti_draw import *

```

```

EDITOR: D
Func Ctl Ops List Type I/O Modul
1:math...
2:random...
3:time...
4:ti_system...
5:ti_plotlib...
6:ti_hub...
7:ti_rover...
8:ti_draw...

```

ti_draw menus

```

EDITOR: D
Shape Control Ver
1:draw_line(x1,y1,x2,y2)
2:draw_rect(x,y,w,h)
3:fill_rect(x,y,w,h)
4:draw_circle(x,y,r)
5:fill_circle(x,y,r)
6:draw_text(x,y,"string")
7:draw_poly(x-list,y-list)
8:fill_poly(x-list,y-list)
9:plot_xy(x,y,sh) sh=1-13

```

```

EDITOR: D
Shape Control Ver
1:clear()
2:clear_rect(x,y,w,h)
3:set_color(r,g,b) 0-255
4:set_pen("size","style")
5:set_window(xmn,xmx,ymn,ymx)
6:show_draw() [clear]

```

```

EDITOR: D
Shape Control Ver
1: -- version 1.0 --
2: ©2021 Texas Instruments
3: All Rights Reserved.

```

Programm-Informationen für die Verwendung von ti_draw:

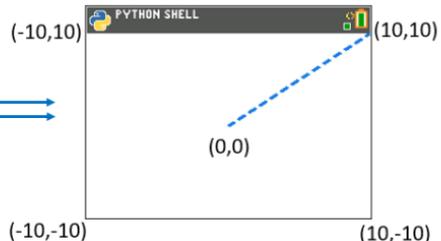
- Verwenden Sie nach der Importanweisung den Befehl `clear()`, um bei Bedarf den Zeichenbereich der Shell zu löschen.
- Programme müssen den Befehl `show_draw()` enthalten, damit das gezeichnete Objekt beim Ausführen des Programms angezeigt wird.
- Mit `draw_rect()`, `draw_circle()` oder `draw_poly()` zeichnen Sie den Rahmen der Form. Die Methoden `fill_rect()`, `fill_circle()` und `fill_poly()` dienen zum Ausfüllen der Innenfläche der jeweiligen Form (je nach Stiftgröße).
- Drücken Sie `[clear]`, um die Zeichnung zu löschen und zur Shell-Eingabeaufforderung zurückzukehren. Hinweis: Das Shell-Protokoll kann mit `[2nd]` `[↑]` und `[2nd]` `[↓]` aufgerufen werden.
- Bitte lesen Sie die Informationen zu den Menüs „Shape“ und „Control“ in der nachstehenden Tabelle genau durch. Ihre Zeichnungen, die Sie mit den Methoden im Menü „Shape“ erstellt haben, hängen von den Methoden im Menü „Control“ ab, wie `set_color()` und `set_pen()`.
- **Koordinatenargumente** sind entweder Bildschirmpixel-Koordinaten oder werden von der Methode `set_window()` vorgegeben.

- ti_draw-Methoden unter Verwendung von `set_window()`-Koordinaten

```

EDITOR: LINEHNDW
PROGRAM LINE 0007
from ti_draw import *
clear()
set_color(25,125,225)
set_pen("medium","dashed")
set_window(-10,10,-10,10)
draw_line(0,0,10,10)
show_draw()_

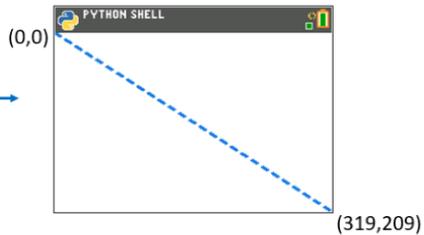
```



- ti_draw-Methoden unter Verwendung von Bildschirmpixel-Koordinaten

```

EDITOR: LINE
PROGRAM LINE 0004
from ti_draw import *
clear()
set_color(25,125,225)
set_pen("medium","dashed")
draw_line(0,0,319,209)
show_draw()
    
```



	Menü „Shape“	Beschreibung
1:	<code>draw_line(x1,y1,x2,y2)</code>	Zeichnet ein Liniensegment zwischen den festgelegten Punkten (x_1,y_1) und (x_2,y_2) .
2:	<code>draw_rect(x,y,w,h)</code>	Zeichnet ein Rechteck mit einer Breite von w Pixeln und einer Höhe von h Pixeln, dessen obere linke Ecke sich an Position (x,y) befindet.
3:	<code>fill_rect(x,y,w,h)</code>	Füllt das Innere eines Rechtecks mit einer Breite von w Pixeln und einer Höhe von h Pixeln aus, dessen obere linke Ecke sich an Position (x,y) befindet.
4:	<code>draw_circle(x,y,r)</code>	Zeichnet einen Kreis mit dem Mittelpunkt bei (x,y) und einem Radius von r Pixeln.
5:	<code>fill_circle(x,y,r)</code>	Zeichnet einen Kreis mit dem Mittelpunkt bei (x,y) und einem Radius von r Pixeln, der mit der festgelegten Farbe ausgefüllt wird (entweder über <code>set_color</code> oder Schwarz, wenn keine Farbe festgelegt wurde).
6:	<code>draw_text(x,y,"string")</code>	Zeichnet die Zeichenkette („string“) als Text auf dem Display, wobei die obere linke Ecke des Texts an Position (x,y) beginnt.
7:	<code>draw_poly(x-list,y-list)</code>	Zeichnet eine Reihe von Linien, die ein Polygon ergeben könnten. Die Linien werden mit der jeweils aktuellen Stiftgröße und Stiftfarbe gezeichnet.
8:	<code>fill_poly(x-list,y-list)</code>	Die Werte für <code>x-list</code> und <code>y-list</code> müssen gleich lange Listenargumente in einer Liste von (x,y) -Eckpunkten sein. Das Polygon wird durch Verbindung der einzelnen Eckpunktpaare gezeichnet und mit der aktuellen Stiftfarbe ausgefüllt.
9:	<code>poly_xy(x,y,sh) sh=1-13</code>	Je nach dem ausgewählten Formenwert

Menü „Shape“	Beschreibung
	<p>(sh) wird eine der nachfolgend aufgelisteten Formen gezeichnet. Dabei dienen die x- und y-Argumente als Mittelpunkt. Die Formen werden mit der aktuellen Stiftfarbe gezeichnet.</p> <p>Form Beschreibung</p> <ol style="list-style-type: none"> 1 Ausgefüllter Kreis mit einem Radius von 2 2 Leerer Kreis mit einem Radius von 2 3 Ausgefülltes Quadrat, 3 x 3 4 Leeres Quadrat, 3 x 3 5 x-Zeichen wird gezeichnet 6 +-Zeichen wird gezeichnet 7 Einzelnes Pixel 8 Ausgefüllter Kreis mit einem Radius von 4 Pixeln 9 Leerer Kreis mit einem Radius von 4 Pixeln 10 Ausgefüllter Kreis mit einem Radius von 6 Pixeln 11 Leerer Kreis mit einem Radius von 6 Pixeln 12 Ausgefüllter Kreis mit einem Radius von 8 Pixeln 13 Leerer Kreis mit einem Radius von 8 Pixeln

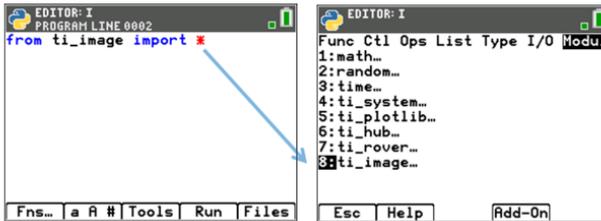
Menü „Control“	Beschreibung
1:	<p><code>clear()</code></p> <p>Löscht den Zeichenbereich in der Shell. <code>clear()</code> muss vor dem Zeichnen angewandt werden, damit der Shell-Zeichenbereich leer ist und die gewünschten Ergebnisse erzielt werden.</p>
2:	<p><code>clear_rect(x,y,w,h)</code></p> <p>Füllt das Innere eines Rechtecks mit einer Breite von w und einer Höhe von h aus, dessen obere linke Ecke sich an Position (x,y) befindet.</p> <p>Weiß ist die Standard-Füllfarbe. Nach dem Einfügen der Methode in den Editor</p>

	Menü „Control“	Beschreibung
		kann die Methode ein fünftes optionales Argument akzeptieren, um eine andere Farbe durch die Verwendung eines Tupels mit (r,g,b)-Werten festzulegen. Ein gültiges (r,g,b)-Tupel enthält Ganzzahlenwerte zwischen 0 und 255.
3:	<code>set_color(r,g,b) 0-255</code>	Legt die Stiftfarbe mithilfe eines (r,g,b)-Tupels fest.
4:	<code>set_pen("size","style")</code>	<p>Legt die Größe („size“) und den Stil („style“) für den Zeichenstift für alle nachfolgenden Zeichnungen fest, bis diese Werte geändert werden.</p> <p>Beim Import von <code>ti_draw</code> stehen für die Größe die Werte „thin“ (dünn), „medium“ (mittel) oder „thick“ (dick) und für den Stil „solid“ (durchgezogen), „dotted“ (gepunktet) oder „dashed“ (gestrichelt) zur Auswahl. Wird kein Wert festgelegt, so sind die Standardwerte „thin“ und „solid“. Der Argumente Helper > hilft beim Ausfüllen der korrekten Argumentzeichenketten.</p> <p>Hinweis: Beim Import des Moduls <code>ti_plotlib</code> stehen als Stilargumente für <code>pen()</code> „solid“, „dot“ oder „dash“ zur Auswahl.</p>
5:	<code>set_window(xmn,xmx,ymn,ymx)</code>	<p>Legt den Zeichenbereich mit den Koordinatenbereichen [xmin,xmax] und [ymin,ymax] fest, wobei (0,0) der Mittelpunkt der Bereiche ist. Hinweis: Liegen die Argumentwerte außerhalb des festgelegten Zeichenbereichs, wird kein Fehler ausgegeben.</p> <p>Wenn <code>set_window(xmin,xmax,ymin,ymax)</code> in einem Programm nicht ausgeführt wird, beträgt die Pixel-Fenstergröße standardmäßig (xmin,xmax,ymin,ymax) = (0,319,0,209). Dabei ist (0,0) als obere linke Eckpixel-Koordinate des Bereichs festgelegt.</p>
6:	<code>show_draw() [clear]</code>	Muss enthalten sein, um die Zeichnung anzuzeigen. Drücken Sie [clear], um die Zeichnung zu löschen und zur Shell-Eingabeaufforderung zurückzukehren.

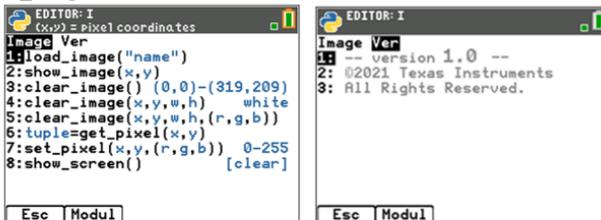
	Menü „Control“	Beschreibung
		Verwenden Sie die Tasten [2nd]  und [2nd]  , um das Shell-Protokoll einzusehen.

[Fns...] ti_image Add-on-Modul

Das ti_image-Modul ist im neuesten CE-Bundle enthalten. Fügen Sie die Importanweisung über [Fns...] > Modul [Add-On] in Ihr Programm ein. Das ti_image-Menü wird dann im Menü [Fns...] > Modul folgendermaßen angezeigt.



ti_image menus



Programm-Informationen für die Verwendung von ti_image:

- Das ti_image-Modul kann für Folgendes verwendet werden:
 - Anzeige der von Ihnen benannten Python-Bild-AppVar, die im Archivspeicher Ihres CE gespeichert ist
 - Anzeige der mit der festgelegten Farbe ausgefüllten Rechtecke an einer Pixelkoordinaten-Position
 - Festlegen oder Abrufen einer Pixelfarbe
 - Löschen der Füllung eines rechteckigen Bildschirmbereichs
 - Löschen des Zeichenbereichs in der Shell auf dem gesamten Bildschirm im Pixelkoordinatenbereich (0,0) bis (319,209)
- Verwenden Sie nach der Importanweisung die Methode clear_image(), um bei Bedarf den Zeichenbereich der Shell zu löschen.
- Die Python-Bild-AppVar ist eine spezielle Python-Bilddatei (*.8xv).
 - Derzeit finden Sie unter <https://resources.t3europe.eu/t3europe-home?country=15&q=images&cHash=d50a2b65ab1b875dfa3ec11bca12154c> eine Bibliothek mit Bildern für den Unterricht.
 - Beim Verwenden einer Python-Bild-AppVar empfiehlt es sich,
 - die Python-Bild-AppVar im Archivspeicher zu speichern. [2nd][mem]
 - die Pixelmaße Ihres Bildes für die Programmierung zu kennen.

- den genauen Namen der Python-Bild-AppVar zu kennen. Sie müssen den Namen in der korrekten Schreibweise mit Groß- und Kleinschreibung eingeben. Bei falsch geschriebenen Python-Bild-AppVar-Namen wird keine Fehlermeldung ausgegeben.
- Die neuesten Versionen von TI Connect™ CE und TI-SmartView™ CE finden Sie unter education.ti.com/84cetupdate.
- Bei (x,y)-Koordinatenargumenten handelt es sich NUR bei ti_image-Methoden um Pixelkoordinaten mit einem Bereich zwischen (0,0) und (319,209). In der nachstehenden Tabelle finden Sie weitere Informationen zu den einzelnen Methoden. Einige Methoden können in verschiedenen Formaten in den Editor eingefügt werden, wenn optionale Argumente angeboten werden.
- Drücken Sie [clear], um die Zeichnung zu löschen und zur Shell-Eingabeaufforderung zurückzukehren. Das Shell-Protokoll kann mit [2nd]  und [2nd]  aufgerufen werden.

	Menü „Control“	Beschreibung
1:	<code>load_image("name")</code>	<p>Lädt einen Namen für eine Python-Bild-AppVar, der für das Programm verwendet werden kann.</p> <p>Der Name des Python-Bildes muss exakt so geschrieben werden wie der Name der Python-Bild-AppVar (Groß-/Kleinschreibung beachten). Hinweis: Es wird keine Fehlermeldung ausgegeben, wenn der AppVar-Name NICHT exakt angegeben wird.</p> <p>Das Python-Bild mit dem festgelegten „name“ wird angezeigt, wenn die Methode <code>show_image(x,y)</code> angewendet wird.</p> <p>Bewährte Vorgehensweisen:</p> <ul style="list-style-type: none"> • Notieren Sie sich die Pixelmaße Ihres Python-Bilds. • Speichertipp: Python-Bild-AppVar-Dateien sollten im Archivspeicher gespeichert werden.
2:	<code>show_image(x,y)</code>	<p>Zeigt das unter <code>load_image("name")</code> festgelegte Bild an.</p> <p>Zeigt das Bild an, wobei sich die obere linke Pixelecke an Position (x,y) des Zeichenbereichs in der Shell befindet. (x,y)-Bildschirmpixel-Koordinaten</p>

	Menü „Control“	Beschreibung
		<p>reichen von (0,0) links oben bis (319,209) rechts unten.</p> <p>Wurde über <code>load_image()</code> kein Bildname angegeben, wird beim Ausführen des Programms eine Fehlermeldung ausgegeben. Wird der Name unter „name“ falsch eingegeben, wird kein Fehler ausgegeben.</p> <p>Mit der Methode <code>show_screen()</code> wird das Bild so lange angezeigt, bis der Benutzer über den Befehl <code>[clear]</code> zur Shell zurückkehrt. Verwenden Sie die Tasten <code>[2nd] [↩]</code> und <code>[2nd] [⏏]</code>, um das Shell-Protokoll einzusehen.</p>
3:	<code>clear_image()</code> (0,0)-(319,209)	<p>Mit der Methode <code>clear_image()</code> ohne Argumente lässt sich der Zeichenbereich der Shell löschen. Der Zeichenbereich wird dann als weißer Bildschirm angezeigt.</p> <p>Die Pixelkoordinaten reichen von (0,0) links oben bis (319,209) rechts unten.</p> <p>Nachdem der gesamte Zeichenbereich mit dieser Methode gelöscht wurde, können Sie sich mit <code>load_image("name")</code> und <code>show_image(x,y)</code> das gewünschte Bild mit dem festgelegten Namen anzeigen lassen.</p> <p>Beachten Sie, dass bei zusätzlicher Verwendung von Methoden des <code>ti_draw</code>-Moduls die Stiftfarbe unter <code>set_pen()</code> auf Schwarz eingestellt wird, wenn die <code>ti_image</code>-Methode <code>clear_image()</code> ausgeführt wird.</p>
4:	<code>clear_image(x,y,w,h) white</code>	<p>Bei einem Rechteck mit einer Breite von <code>w</code> Pixeln und einer Höhe von <code>h</code> Pixeln, bei dem sich die Pixelkoordinate für die linke obere Ecke an Position <code>(x,y)</code> befindet, wird mit dieser Methode die Innenfläche des Rechtecks „gelöscht“ und weiß dargestellt.</p>
5:	<code>clear_image(x,y,w,h,(r,g,b))</code>	<p>Bei einem Rechteck mit einer Breite von <code>w</code> Pixeln und einer Höhe von <code>h</code> Pixeln, bei</p>

	Menü „Control“	Beschreibung
		dem sich die Pixelkoordinate für die linke obere Ecke an Position (x,y) befindet, wird mit dieser Methode die Innenfläche des Rechtecks „gelöscht“ und in der entsprechenden RGB-Farbe des festgelegten (r,g,b)-Tupels dargestellt.
6:	<code>tuple=get_pixel(x,y)</code>	Gibt die RGB-Werte des Pixels an der Pixelkoordinate (x,y) als (r,g,b)-Tupel aus.
7:	<code>set_pixel(x,y,(r,g,b))</code>	Stellt die Farbe des Pixels an der Pixelkoordinate (x,y) auf die RGB-Farbe ein, die unter (r,g,b) festgelegt wurde.
8:	<code>show_screen()</code> [clear]	<p>Diese Methode dient dazu, dass die Zeichnung weiterhin auf dem Bildschirm angezeigt wird, wenn das Modul <code>ti_image</code> verwendet wird.</p> <p>Wenn nach jedem Aufruf von <code>show_screen()</code> die Taste [clear] gedrückt wird, wird das Programm weiterhin ausgeführt, bis der Bildschirm zur Shell-Eingabeaufforderung zurückkehrt.</p> <p>Verwenden Sie die Tasten [2nd] <input type="checkbox"/> und [2nd] <input type="checkbox"/>, um das Shell-Protokoll einzusehen. Weitere Shell-Optionen finden Sie unter Shell > [Tools].</p>

Meldungen in der Python-App

Während einer Python-Sitzung können mehrere Meldungen angezeigt werden. Einige ausgewählte Meldungen sind in der Tabelle aufgeführt. Bitte folgen Sie den Anweisungen auf dem Bildschirm und navigieren Sie je nach Bedarf mit [Quit], [Esc] oder [Ok].

Speicherverwaltung

Der verfügbare Speicher der Python-Umgebung beträgt maximal 100 Python-Programme (PY AppVars) oder 50K Speicherplatz. Die Module, die in dieser Python-Version mit der App gebündelt sind, teilen sich mit allen Dateien den gleichen Speicherplatz.



Verwenden Sie [2nd] [quit], um die App zu beenden

Sie werden aufgefordert, zu bestätigen, dass Sie die App beenden möchten. Durch Beenden der App wird die Python-Sitzung gestoppt. Wenn Sie die Python-App erneut ausführen, werden Ihre Python AppVar-Programme und Module synchronisiert. Die Shell wird reinitialisiert.



Sie drücken im Dateimanager bei einem ausgewählten Python-Programm auf [del] oder Sie wählen in **File Manager>Manage 2:Delete Program...**

Im angezeigten Dialogfeld können Sie wählen, das Programm zu löschen oder zum Dateimanager zurückzukehren.



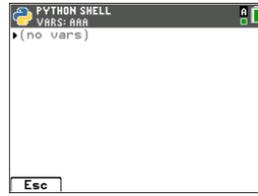
Sie versuchen, ein neues Python-Programm zu erstellen oder ein Python-Programm zu duplizieren, das entweder im RAM oder im Archiv Ihres CE vorhanden oder für den Prüfungsmodus deaktiviert ist. Bitte geben Sie einen anderen Namen ein.



Sie versuchen, von der Shell zum Editor zu navigieren, der Editor ist jedoch leer. Bitte wählen Sie eine geeignete Option für Ihre Arbeit aus.



Wenn Sie ein Python-Programm ausführen, werden die definierten Variablen des zuletzt ausgeführten Programms im **Menü Shell>Tools> 4:Vars...** aufgelistet und stehen für die Verwendung in der Shell zur Verfügung. Wenn keine Variablen angezeigt werden, müssen Sie Ihr Programm möglicherweise erneut ausführen.



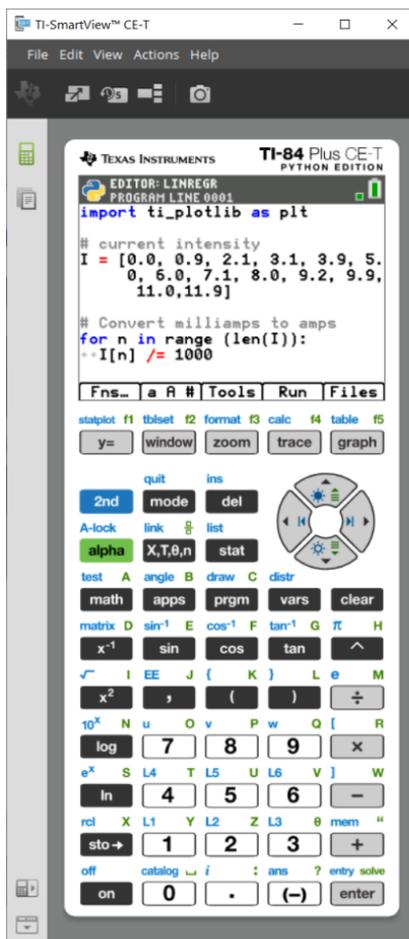
Verwendung von TI-SmartView™ CE-T und der Python-Umgebung

Dieses Handbuch bezieht sich auf die neueste Version von TI-SmartView™ CE-T. Aktualisieren Sie TI-SmartView™ CE-T unter education.ti.com/84cetupdate auf die neueste Version.

Das Update enthält das neueste TI-84 Plus CE-T Python Edition Emulator-Betriebssystem mit der neuesten Python-App. Es enthält die aktualisierten Module von time, ti_system, ti_plotlib, ti_rover* und ti_hub*.

Führen Sie die Python-App im TI-84 Plus CE-T Python Edition Emulator aus.

- Die Python-App bietet:
 - Dateimanager
 - Editor
 - Ausführung Ihres Python-Programms in der Shell*



Hub-/Rover-Programme

- Wenn im CE-Emulator die Python-App läuft, können Sie ti_hub/ti_rover-Python-Programme erstellen.
 - * **Hinweis:** Es besteht keine Konnektivität zwischen TI-SmartView™ CE-T und TI-Innovator™ Hub bzw. TI-Innovator™ Rover. Programme können erstellt und dann auf dem CE-Taschenrechner ausgeführt werden.
- Beenden Sie die Python-App, um die Übertragung der Python-AppVar(s) aus dem Emulator vorzubereiten. Beim nächsten Schritt sollte der Emulator nicht damit beschäftigt sein, eine App oder ein Programm auszuführen.

- Wechseln Sie zum Arbeitsbereich Emulator-Explorer und senden Sie das/die Programm(e) an den Computer.
- Verwenden Sie für die TI-Innovator™ Hub/TI-Innovator™ Rover-Erfahrung TI Connect™ CE, um die Python AppVars vom Computer an den CE-Taschenrechner zu senden.

Hinweis: Um ein laufendes Python-Programm in der Shell zu unterbrechen, z. B. wenn sich ein Programm in einer Endlosschleife befindet, drücken Sie **[on]**. Alternativ können Sie **[Tools] [zoom] > 6:New Shell** drücken, um ein laufendes Programm anzuhalten.

Zur Erinnerung: Für alle Computer/TI-Python-Umgebungen: Nach dem Erstellen eines Python-Programms in einer Python-Entwicklungsumgebung auf dem Computer überprüfen Sie bitte die Ausführung Ihres Programms auf dem Rechner/Emulator in der TI-Python-Umgebung. Nehmen Sie ggf. Änderungen am Programm vor.

Remote-Tastatur der SmartPad CE-App

- Wenn die SmartPad CE-App auf Ihrem angeschlossenen CE ausgeführt wird, verhält sich dieses wie eine Remote-[Tastatur](#) einschließlich der speziellen Tastaturbelegung, die beim Ausführen der Python-App angeboten wird.

Arbeitsbereich Emulator-Explorer

- Bitte beenden Sie die Python-App, damit der Emulator nicht beschäftigt ist, wenn Sie auf den vollen Funktionsumfang des Arbeitsbereichs Emulator-Explorer zugreifen.
- Umwandlungen zwischen program.py und PY AppVar sind erlaubt. Dies ähnelt dem Einsatz von TI Connect™ CE beim Senden von Programmen an den angeschlossenen CE-Taschenrechner.
- Zum Senden einer program.py-Datei, die in einer anderen Python-Umgebung erstellt wurde, muss die PY AppVar bearbeitet werden, damit sie in TI-Python wie erwartet läuft. Verwenden Sie den Editor der Python-App, um die erforderlichen Änderungen für die einzelnen Module wie ti_plotlib, ti_system, ti_hub und ti_rover vorzunehmen.

Datenimport-Assistent

- *.csv-Datendateien, die wie im Dialogfeld des Assistenten angegeben formatiert sind, werden in CE-Listenvariablen konvertiert. Anschließend können Methoden in ti_system verwendet werden, um Listen zwischen dem Emulator CE-Betriebssystem und der Python-App auszutauschen. Diese Funktion ähnelt dem Datenimport-Assistenten in TI Connect™ CE.
- Wenn Dezimalzahlen in der *.csv-Datei mit Verwendung eines Kommas dargestellt werden, wird die Datei nicht vom Datenimport-Assistenten konvertiert. Bitte überprüfen Sie die Zahlenformatierung Ihres Computer-Betriebssystems und konvertieren Sie das *.csv-Format, um die Dezimalpunktdarstellung zu verwenden. Beispiel: Die CE-Taschenrechnerliste und der Matrixeditor verwenden das Zahlenformat 12.34 und nicht 12,34.

Verwenden von TI Connect™ CE zum Konvertieren von Python-Programmen

Bitte aktualisieren Sie auf TI Connect™ CE, um die neuesten Funktionen zu erhalten, einschließlich der Konvertierung von *.py-Programmen in eine PY AppVar im CE-Taschenrechnerdateiformat.

Detailliertere Informationen zum CE Taschenrechner, TI-SmartView™ CE-T und TI Connect CE finden Sie im [TI-84 Plus CE-T e-Guide](#).

Was ist die Python-Programmierungsumgebung?

TI-Python basiert auf CircuitPython, einer Python-Variante, die für kleine Mikrocontroller entwickelt wurde. Die ursprüngliche CircuitPython-Implementierung wurde für die Verwendung durch TI angepasst.

Die interne Speicherung von Zahlen zur Berechnung in dieser Variante von Circuit Python erfolgt in binären Fließkommazahlen mit begrenzter Genauigkeit und kann daher nicht alle möglichen Dezimalwerte exakt darstellen. Die bei der Speicherung dieser Werte auftretenden Abweichungen von den tatsächlichen Dezimaldarstellungen können bei nachfolgenden Berechnungen zu unerwarteten Ergebnissen führen.

- **Bei Fließkommazahlen** – Es werden bis zu 16 signifikante Genauigkeitsziffern angezeigt. Intern werden die Werte mit einer Genauigkeit von 53 Bit gespeichert, was in etwa 15-16 Dezimalstellen entspricht.
- **Bei Ganzzahlen** – Die Größe von Ganzzahlen ist nur durch den zum Zeitpunkt der Berechnungen verfügbaren Speicher begrenzt.

Im TI-84 Plus CE-T Python Edition enthaltene Module

- [Built-in \(integriert\)](#)
- [math module](#)
- [random module](#)
- [time](#)
- [ti_system](#)
- [ti_plotlib](#)
- [ti_hub](#)
- [ti_rover](#)

Hinweis: Wenn Sie Python-Programme besitzen, die in anderen Python-Entwicklungsumgebungen erstellt wurden, passen Sie Ihr(e) Programm(e) bitte an die TI-Python-Lösung an. Die Module können in einem Programm andere Methoden, Argumente und Methoden-Reihenfolgen verwenden als die Module `ti_system`, `ti_plotlib`, `ti_hub` und `ti_rover` modules. Achten Sie bei der Verwendung aller Python-Versionen und Python-Module grundsätzlich auf Kompatibilität.

Beim Übertragen von Python-Programmen von einer Nicht-TI-Plattform auf eine TI-Plattform ODER von einem TI-Produkt auf ein anderes:

- Python-Programme, die Kernsprachenfunktionen und Standardbibliotheken (`math`, `random` usw.) verwenden, können ohne Änderungen importiert oder exportiert werden.

Hinweis: Die Begrenzung der Listenlänge beträgt 100 Elemente.

- Programme, die plattformspezifische Bibliotheken – `matplotlib` (für PC), `ti_plotlib`, `ti_system`, `ti_hub` usw. – für TI-Plattformen verwenden, müssen bearbeitet werden, bevor sie auf einer anderen Plattform ausgeführt werden können.

- Dies kann sogar zwischen TI-Plattformen erforderlich sein.

Wie bei jeder Python-Version müssen Sie Importe, wie z. B. aus `math import *`, einbinden, um alle Funktionen, Methoden oder Konstanten verwenden zu können, die im `math`-Modul enthalten sind. Um beispielsweise die Funktion `cos()` auszuführen, importieren Sie mit „import“ das `math`-Modul zur Verwendung.

Siehe [KATALOG-Liste](#).

Beispiel:

```
>>>from math import *
>>>cos(0)
1.0
```

Alternatives Beispiel:

```
>>>import math
>>>math.cos(0)
1.0
```

Verfügbare Module können in der Shell mit folgendem Befehl angezeigt werden:

```
>>> help("modules")
__main__ sys gc
random time array
math builtins collections
```

Der Inhalt von Modulen kann in der Shell wie gezeigt mit „import module“ und „dir (module)“ angezeigt werden.

Nicht alle Modulinhalte werden in den Schnelleinfüge-Menüs wie z. B. [Fns...] oder **[2nd]** [catalog] angezeigt.

Inhalte ausgewählter Module und Schlüsselwörter

Eine Liste der in dieser Version enthaltenen Module finden Sie in:

[Anhang: Ausgewählte Inhalte der in TI-Python integrierten Funktionen, Schlüsselwörter und Module](#)

Zur Erinnerung: Für alle Computer/TI-Python-Umgebungen: Nach dem Erstellen eines Python-Programms auf dem Computer überprüfen Sie bitte die Ausführung Ihres Programms auf dem Taschenrechner in der TI-Python-Umgebung. Nehmen Sie ggf. Änderungen am Programm vor.

Diese Bildschirme zeigen die Modulinhalte für Mathematik und Zufall an.



```
PYTHON SHELL
>>> import math
>>> dir(math)
['_name__', 'e', 'pi', 'sqrt',
'pow', 'exp', 'log', 'cos', 'sin',
'tan', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'copysign',
'fabs', 'floor', 'fmod', 'frexp',
'ldexp', 'modf', 'isfinite', 'i
sinf', 'isnan', 'trunc', 'radian
s', 'degrees']
>>> |
```

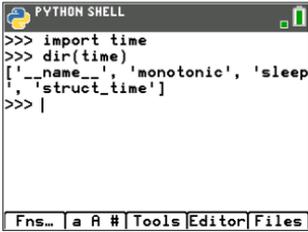
math module



```
PYTHON SHELL
>>> import random
>>> dir(random)
['_name__', 'seed', 'getrandbit
s', 'randrange', 'randint', 'cho
ice', 'random', 'uniform']
>>> |
```

random module

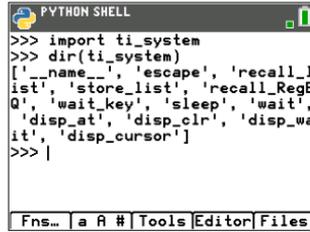
Diese Bildschirme zeigen den Modulinhalt für `time` und `ti_system` an.



```
PYTHON SHELL
>>> import time
>>> dir(time)
['__name__', 'monotonic', 'sleep', 'struct_time']
>>> |
```

Fns... a A # Tools Editor Files

`time`



```
PYTHON SHELL
>>> import ti_system
>>> dir(ti_system)
['__name__', 'escape', 'recall_list', 'store_list', 'recall_RegEQ', 'wait_key', 'sleep', 'wait', 'disp_at', 'disp_clr', 'disp_wait', 'disp_cursor']
>>> |
```

Fns... a A # Tools Editor Files

`ti_system`

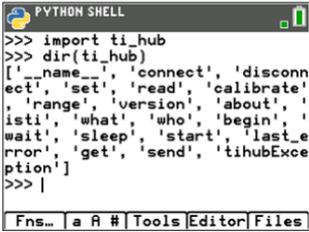
Diese Bildschirme zeigen den Modulinhalt für `ti_plotlib` an.



```
PYTHON SHELL
>>> import ti_plotlib
>>> dir(ti_plotlib)
['lin_reg', 'strtest', 'escape',
 '_excpt', 'text_at', '_clipseg',
 'show_plot', 'tilocal', 'pen',
 'sys', 'xmin', 'ymax', 'yscl',
 '_xy', '_rdelta', '_ydelta', 'scatter',
 'a', '_pencolor', 'write', 'b',
 '_xytest', 'window', '_mark',
 'line', 'monotonic', '_numtest',
 'ymin', 'tiplotlibException',
 'labels', 'cls', 'sqrt', 'xscl',
 'axes', 'grid', '_sema', '_pensize',
 'plot', 'isnan', 'color',
 'title', '_xdelta', '_penstyle',
 '__name__', 'copysign', 'gr',
 'xmax', 'sleep', 'auto_window']
>>> |
Fns... | a A # |Outils|Éditer|Script
```

`ti_plotlib`

Dieser Bildschirm zeigt den Modulinhalt für ti_hub an.



```
PYTHON SHELL
>>> import ti_hub
>>> dir(ti_hub)
['_name_', 'connect', 'disconnect', 'set', 'read', 'calibrate', 'range', 'version', 'about', 'isti', 'what', 'who', 'begin', 'wait', 'sleep', 'start', 'last_error', 'get', 'send', 'tihubException']
>>> |
Fns... | a R # | Tools | Editor | Files
```

ti_hub

Diese Bildschirme zeigen den Modulinhalt für `ti_rover` an.



```
PYTHON SHELL
>>> import ti_rover
>>> dir(ti_rover)
['motor_right', 'to_angle', 'to_xy', 'red_measurement', 'rvmove
ment', 'gray_measurement', 'exc
pt', 'pathlist_time', 'waypoint_
prev', 'ti_hub', 'waypoint_eta',
'to_polar', 'grid_unit', 'col
or_off', 'path_clear', 'rv', 'g
reen_measurement', 'motors', 'wa
ypoint_time', 'backward', 'color
_blink', 'motor_left', 'waypoint
_heading', 'motor', 'gyro_measur
ement', 'wait_until_done', 'enc
oders_gyro_measurement', 'pathli
st_distance', 'position', 'blue_
measurement', 'forward', 'waypoi
nt_distance', 'grid_origin', 're
sume', 'path_done', 'disconnect_
rv', 'backward_time', 'zero_gyro',
'rv_connected', 'stop', 'sta
y', 'waypoint_xythdrn', 'ranger_
measurement', 'left', 'pathlist_
cmdnum', 'waypoint_y', 'waypoint
_x', 'pathlist_y', 'pathlist_x',
'__name__', 'right', 'color_rgb',
'pathlist_revs', 'color_mesu
rement', 'pathlist_heading', 'fo
rward_time', 'waypoint_revs']
>>> |
Fns... a A # Tools Editor Files
```

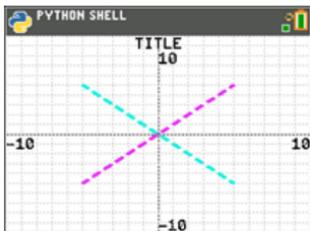
`ti_rover`

Beispielprogramme

Verwenden Sie die folgenden Beispielprogramme, um sich mit den Methoden aus dem Abschnitt [Referenz](#) vertraut zu machen. Diese Beispiele enthalten auch mehrere TI-Innovator™ Hub- und TI-Innovator Rover™-Programme, die Ihnen den Einstieg in TI-Python erleichtern sollen.

COLORLIN

```
import ti_plotlib as plt
plt.cls()
plt.window(-10,10,-10,10)
plt.axes("on")
plt.grid(1,1,"dot")
plt.title("TITLE")
plt.pen("medium","solid")
plt.color(28,242,221)
plt.pen("medium","dash")
plt.line(-5,5,5,-5,"")
plt.color(224,54,243)
plt.line(-5,-5,5,5,"")
plt.show_plot()
```



Drücken Sie **clear**, um die Shell-Eingabeaufforderung anzuzeigen.

REGEQ1

Stellen Sie eine Regressionsgleichung auf, bevor Sie das Python-Programm in der Python-App ausführen. Ein Beispiel wäre, zuerst zwei Listen in das CE-Betriebssystem einzugeben. Berechnen Sie dann zum Beispiel [stat] CALC 4:LinReg(ax+b) für Ihre Listen. So wird die Regressionsgleichung im Betriebssystem in RegEQ gespeichert. Über das folgende Programm wird RegEQ in der Python-Umgebung aufgerufen.

```
# Example of recall_RegEQ()
from ti_system import *

reg=recall_RegEQ()
print(reg)
x=float(input("Input x = "))
print("RegEQ(x) = ",eval(reg))
```

LINREGR (Im CE-Bundle enthalten)

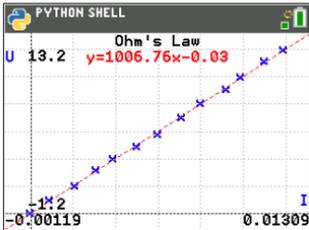
```
import ti_plotlib as plt

# current intensity
I = [0.0, 0.9, 2.1, 3.1, 3.9, 5.0, 6.0, 7.1, 8.0, 9.2, 9.9, 11.0, 11.9]

# voltage
for n in range (len(I)):
    I[n] /= 1000

# la tension
U = [0, 1, 2, 3.2, 4, 4.9, 5.8, 7, 8.1, 9.1, 10, 11.2, 12]

plt.cls()
plt.auto_window(I,U)
plt.pen("thin", "solid")
plt.axes("on")
plt.grid(.002, 2, "dot")
plt.title("Ohm's Law")
plt.color (0,0,255)
plt.labels("I", "U", 11, 2)
plt.scatter(I,U, "x")
plt.color (255,0,0)
plt.pen("thin", "dash")
plt.lin_reg(I,U, "center", 2)
plt.show_plot()
plt.cls()
a=plt.a
b=plt.b
print ("a =", round(plt.a, 2))
print ("b =", round(plt.b, 2))
```



Drücken Sie , um die Shell-Eingabeaufforderung anzuzeigen.

GRAPH (Im CE-Bundle enthalten)

```
import ti_plotlib as plt
#Nach Ausführen des Programms [clear] drücken, um das Diagramm zu
löschen und zur Shell zurückzukehren.

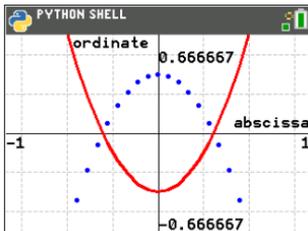
def f(x):
**return 3*x**2-.4

def g(x):
**return -f(x)

def plot(res,xmin,xmax):
**#setup plotting area
**plt.window(xmin,xmax,xmin/1.5,xmax/1.5)
**plt.cls()
**gscale=5
**plt.grid((plt.xmax-plt.xmin)/gscale*(3/4),(plt.ymax-
plt.ymin)/gscale,"dash")
**plt.pen("thin","solid")
**plt.color(0,0,0)
**plt.axes("on")
**plt.labels("abscisse"," ordonnee",6,1)
**plt.pen("medium","solid")

# plot f(x) and g(x)
dX=(plt.xmax -plt.xmin)/res
x=plt.xmin
x0=x
**for i in range(res):
***plt.color(255,0,0)
***plt.line(x0,f(x0),x,f(x),"")
***plt.color(0,0,255)
***plt.plot(x,g(x),"o")
***x0=x
***x+=dX
**plt.show_plot()

#plot(resolution,xmin,xmax)
plot(30,-1,1)
# Ein Diagramm mit parameters(resolution,xmin,xmax) erstellen
# Nach Löschen des ersten Diagramms Taste [var] drücken. Die Funktion
plot() function ermöglicht das Ändern der Anzeigeeinstellungen
(resolution,xmin,xmax).
```



Drücken Sie `clear`, um die Shell-Eingabeaufforderung anzuzeigen.

DASH1 – TI-Innovator™ Hub-Beispielprogramm

Siehe: [\[Fns...\]>Modul: ti_hub module](#)

```
from ti_system import *
import brightns
import ti_plotlib as plt
from time import *
plt.cls()
plt.color(0,0,255)
plt.text_at(2,"Monitoring Hub","center")
plt.text_at(3,"Brightness Sensor","center")
plt.color(255,0,0)
plt.text_at(12,"Press [clear] to quit ","right")
t0=monotonic()
plt.color(0,0,0)
while not escape():
    *I=brightns.measurement()
    *I=round(I,1)
    *tf=monotonic()
    *plt.color(0,0,0)
    *tm=round(tf-t0,1)
    *msg="Time = %.1f sec" % tm
    *plt.text_at(6,msg,"center")
    *msg="Brightness = %.1f %%" %I
    *plt.text_at(7,msg,"center")
    *sleep(1)
```



The screenshot shows a window titled "EDITOR: DASH1" with a subtitle "PROGRAM LINE 0001". The code is displayed in a monospaced font with syntax highlighting: keywords in blue, strings in green, and comments in red. The code is identical to the one shown in the text block above. At the bottom of the window, there is a menu bar with the following items: "Fns...", "a", "A", "#", "Tools", "Run", and "Files".

ROVER – TI-Innovator™ Rover-Beispielprogramm

Siehe: [\[Fns...\]>Modul ti_rover module](#)

```
from ti_system import *
import ti_rover as rv
disp_clr()
disp_cursor(0)
disp_at(6,"Press [clear] to stop","center")
rv.forward(20)
while not escape():
  **a=rv.ranger_measurement()
  **if a<0.2:
  ***rv.color_rgb(255,0,0)
  ***rv.stop()
  **else:
  ***rv.color_rgb(0,255,0)
  ***rv.resume()
rv.stop()
disp_clr()
rv.color_rgb(0,0,255)
sleep(1)
rv.color_rgb(0,0,0)
```



The screenshot shows a window titled "EDITOR: ROVER" with "PROGRAM LINE 0001" at the top. The code is displayed in a monospaced font with syntax highlighting. The code is identical to the one shown in the previous block. At the bottom of the window, there is a menu bar with the following items: "Fns...", "a A #", "Tools", "Run", and "Files".

BLNKSND – TI-Innovator™ Hub-Beispielprogramm

Siehe: [\[Fns...\]>Modul: ti_hub module](#)



```
EDITOR: BLNKSND
PROGRAM LINE 0001
# ti_hub Module menues
from ti_system import *
import color
import sound
for i in range(1,5):
  --color.rgb(i**2,i**3,i**4-1)
  --color.blink(1,2)
  --sleep(2)
  --sound.tone((i**3+250)/3,.5)
  --sleep(2)
```

Fns... a A # Tools Run Files

SQUARE – TI-Innovator™ Rover-Beispielprogramm

Siehe: [\[Fns...\]>Modul ti_rover module](#)



```
EDITOR: SQUARE
PROGRAM LINE 0001
# ti_rover Module menues_
import ti_rover as rv
for i in range(1,5):
  rv.forward(3)
  rv.left(90)
```

Fns... a A # Tools Run Files

STOP_GO - Sample ti_draw, ti_image, time Program

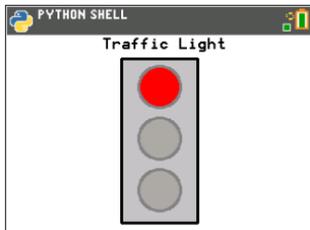
See: [\[Fns...\]>Modul \[Add-On\]](#)

```
from ti_draw import *
from ti_image import *
from time import *
clear()
# Pixel screen upper left (0,0) to (319,209)
draw_text(100,20,"Traffic Light")
set_pen("medium","solid")

draw_rect(120,25,80,175)
set_color(192,192,192)
fill_rect(120,25,80,175)
set_color(128,128,128)
draw_circle(160,55,22)
draw_circle(160,110,22)
draw_circle(160,165,22)

def off(x,y):
    **set_color(169,169,169)
    **fill_circle(x,y,22)
    **set_color(128,128,128)
    **draw_circle(x,y,22)

for i in (1,20,1):
    # Green
    **set_color(51,165,50)
    **fill_circle(160,165,22)
    **sleep(3)
    **off(160,165)
    # Yellow
    **set_color(247,239,10)
    **fill_circle(160,110,22)
    **sleep(2)
    **off(160,110)
    # Red
    **set_color(255,0,0)
    **fill_circle(160,55,22)
    **sleep(3)
    **off(160,55)
    **show_draw()
```



Referenz-Leitfaden für die TI-Python Umgebung

Die Python-App enthält Menüs mit Funktionen, Klassen, Bedienelementen, Operatoren und Schlüsselwörtern zum schnellen Einfügen in den Editor oder die Shell. Die folgende Referenztabelle enthält die Auflistung der Funktionen in [\[2nd\] \[catalog\]](#), wenn die App ausgeführt wird. Eine vollständige Aufstellung der in dieser Version von Python verfügbaren Funktionen, Klassen, Operatoren und Schlüsselwörter finden Sie unter [„Ausgewählte Inhalte der in TI-Python integrierten Funktionen, Schlüsselwörter und Module“](#).

Diese Tabelle ist nicht als erschöpfende Liste der in diesem Angebot verfügbaren Python-Funktionen gedacht. Andere Funktionen, die in diesem Python-Angebot unterstützt werden, können über die ALPHA-Tasten der Tastatur eingegeben werden.

Die meisten Beispiele in dieser Tabelle werden an der Shell-Eingabeaufforderung (>>>) ausgeführt.

KATALOG-Liste

Alphabetische Auflistung

- A
- B
- C
- D
- E
- F
- G
- H
- I
- L
- M
- N
- O
- P
- R
- S
- T
- U
- W
- X
- Y
- Sonderzeichen

A

#

Trennzeichen

[2nd](#) [\[catalog\]](#)

Syntax: #Ihr Kommentar zu Ihrem Programm.

[a A #]

Beschreibung: In Python beginnt ein Kommentar mit dem Hashtag-Zeichen (#) und reicht bis zum Ende der Zeile.

Beispiel:

```
#Kurze Erklärung des Codes.
```

%

Operator

[2nd](#) [\[catalog\]](#)

Syntax: x%y oder x % y

Beschreibung: Gibt den Rest von x/y zurück. Wird bevorzugt verwendet, wenn x und y ganze Zahlen sind.

[a A #]

Beispiel:

```
>>>57%2  
1
```

Siehe auch `fmod(x,y)`.

//

Operator

[2nd](#) [\[catalog\]](#)

Syntax: x//y oder x // y

[a A #]

Beschreibung: Gibt die floor-Division (Division, die jeden Rest verwirft) von x/y.

Beispiel:

```
>>>26//7  
3  
>>>65,4//3  
21.0
```

[a A #]

Beschreibung: Starten der [a A #]-Zeichenpalette.

Diese enthält Zeichen mit Akzenten, wie z. B. ç à â è é ê ë ì
ĩ ô õ ù û

Die [a A #]-
Verknüpfung
wird im
Editor oder
in der Shell
bei `window`
angezeigt

a gradient; slope

Modul: `ti_plotlib`

`2nd` [catalog]

Syntax: `plt.a gradient; slope`

[Fns...]>Modul
oder `math`
5:ti_plotlib...>
Properties
5:a

Beschreibung: Nachdem `plt.linreg()` in einem Programm
zuletzt ausgeführt wurde, werden die berechneten
Werte von `slope`, `a`, und `intercept`, `b`, in `plt.a` und `plt.b`
gespeichert.

Standardwerte: = 0,0

Beispiel:

Siehe Beispielprogramm: [LINREGR](#).

Importbefehle
finden Sie
unter `2nd`
[catalog] oder
im `ti_plotlib`
Setup-Menü.

abs()

Modul: Built-in

`2nd` [catalog]

Syntax: `abs(x)`

Beschreibung: Gibt den Absolutwert einer Zahl zurück.
In dieser Version kann das Argument eine Ganzzahl
oder eine Fließkommazahl sein.

Hinweis:
`fabs()`
ist eine Funktion
im `math`-Modul.

Beispiel:

```
>>>abs(-35.4)
35.4
```

acos()

Modul: math

[\[sin\]](#) 7:acos()

Syntax: acos(x)

Beschreibung: Gibt den Bogenkosinus von x im Bogenmaß (Radian) zurück

[\[2nd\]](#) [\[catalog\]](#)

Beispiel:

```
>>>from math import *
>>>acos(1)
0.0
```

[Fns...] Modul
1:math... > Trig
7:acos()

Alternatives Beispiel: [Tools] > 6:New Shell

```
>>>import math
>>>math.acos(1)
0.0
```

Importbefehle
finden Sie
unter
[\[2nd\]](#) [\[catalog\]](#).

and

Schlüsselwort

[\[2nd\]](#) [\[test\]](#)

Syntax: x and y

Ops 8:and

Beschreibung: Kann True oder False zurückgeben. Gibt "x" zurück, wenn "x" False ist, anderenfalls "y". Wird mit Leerzeichen vor und hinter dem „and“ eingefügt. Kann nach Bedarf bearbeitet werden.

[Fns...] > Ops
8:and

Beispiel:

```
>>>2<5 and 5<10
True
>>>2<5 and 15<10
False
>>>{1} and 3
3
>>>0 and 5 < 10
0
```

[\[2nd\]](#) [\[catalog\]](#)

[a A #]

.append(x)

Modul: Built-in

[2nd](#) [\[list\]](#)

Syntax: listname.append(item)

List
6: .append(x)

Beschreibung: Die Methode append() hängt ein Element an eine Liste an.

Beispiel:

```
>>>listA = [2,4,6,8]
>>>listA.append(10)
>>>print(listA)
[2,4,6,8,10]
```

[2nd](#) [\[catalog\]](#)

[Fns...] > List
6: .append(x)

as

Schlüsselwort

[2nd](#) [\[catalog\]](#)

Beschreibung: Verwenden Sie „as“, um beim Importieren eines Moduls einen Alias zu erstellen. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

asin()

Modul: math

[sin](#) 6:asin()

Syntax: asin()

Beschreibung: Gibt den Bogensinus von x im Bogenmaß (Radian) zurück

[2nd](#) [\[catalog\]](#)

Beispiel:

```
>>>from math import *
>>>asin(1)
1.570796326794897
```

[Fns...] > Modul
1:math... > Trig
6:asin()

Alternatives Beispiel:

```
>>>import math
>>>math.asin(1)
1.570796326794897
```

Importbefehle
finden Sie
unter
[2nd](#) [\[catalog\]](#).

assert

Schlüsselwort

[2nd](#) [\[catalog\]](#)

Beschreibung: Verwenden Sie „assert“, um eine Bedingung in Ihrem Code zu testen. Ergibt „None“; anderenfalls wird bei der Ausführung des Programms ein AssertionError angezeigt.

atan()

Modul: math

[\[sin\]](#) 8:atan()

Syntax: atan(x)

Beschreibung: Gibt die Bogentangente von x im Bogenmaß (Radian) zurück.

[Fns...]>Modul
1:math... > Trig
8 :atan()

Beispiel:

```
>>>from math import *  
>>>atan(1)*4  
3.141592653589793
```

[\[2nd\]](#) [\[catalog\]](#)

Alternatives Beispiel:

```
>>>import math  
>>>math.atan(1)*4  
3.141592653589793
```

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#).

atan2(y,x)

Modul: math

[\[sin\]](#) 9:atan2()

Syntax: atan2(y,x)

Beschreibung: Gibt die Bogentangente von y/x im Bogenmaß (Radian) zurück. Das Ergebnis wird in [-pi, pi] angegeben.

[Fns...] > Modul
1:math... > Trig
9:atan2()

Beispiel:

```
>>>from math import *  
>>>atan2(pi,2)  
1.003884821853887
```

[\[2nd\]](#) [\[catalog\]](#)

Alternatives Beispiel:

```
>>>import math  
>>>math.atan2(math.pi,2)  
1.003884821853887
```

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#).

auto_window(xlist,ylist)

Modul: ti_plotlib

[2nd] [catalog]

Syntax: plt.auto_window(xlist,ylist)

[Fns...]>Modul

oder **[math]**

5:ti_plotlib...>

Setup

5:auto_window

()

Beschreibung: Autoskaliert das Plotting-Fenster so, dass es den Datenbereichen in xlist und ylist entspricht, die im Programm vor auto_window() angegeben wurden.

Hinweis: max(list) - min(list) > 0.00001

Beispiel:

Importbefehle

finden Sie unter

[2nd] [catalog]

oder im Menü

ti_plotlib Setup.

Siehe Beispielprogramm: [LINREGR](#).

axes("mode")

Modul: ti_plotlib

[\[2nd\]](#) [\[catalog\]](#)

Syntax: plt.axes("mode")

[Fns...]>Modul
oder [\[math\]](#)
5:ti_plotlib...>
Setup
6:axes()

Beschreibung: Zeigt Achsen im angegebenen Fenster im Plotting-Bereich an.

Argument:

Optionen im Argument "mode":

"off"	keine Achsen
"on"	Achsen+Achsenbezeichnungen
"axes"	nur Achsen
"windows"	nur Fensterbeschriftungen

Importbefehle finden Sie unter [\[2nd\]](#) [\[catalog\]](#) oder im Menü ti_plotlib Setup.

plt.axes() verwendet die aktuelle Zeichenstift-Farbeinstellung. Um sicherzustellen, dass plt.axes() immer wie erwartet gezeichnet werden, verwenden Sie plt.color() VOR plt.axes(), damit die gewünschten Farben verwendet werden.

Beispiel:

Siehe Beispielprogramm [LINREGR](#).

B

b y= intercept

Modul: ti_plotlib

[\[2nd\]](#) [\[catalog\]](#)

Syntax: plt.b **y= intercept**

[Fns...]>Modul
oder [\[math\]](#)
5:ti_plotlib...>
Properties
6:b

Beschreibung: Nachdem plt.linreg() in einem Programm ausgeführt wurde, werden die berechneten Werte von slope, a, und intercept, b, in plt.a und plt.b gespeichert.

Standardwerte: = 0,0

Beispiel:

Siehe Beispielprogramm [LINREGR](#).

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#)
oder im
ti_plotlib Setup-
Menü.

bin(integer)

Modul: Built-in

[\[2nd\]](#) [\[catalog\]](#)

Syntax: bin(**integer**)

Beschreibung: Zeigt das Binärformat des Arguments „integer“ an.

Weitere Einzelheiten finden Sie in der Python-Dokumentation.

Beispiel:

```
>>> bin(2)
'0b10'
>>> bin(4)
'0b100'
```

break

Schlüsselwort

[\[2nd\]](#) [\[catalog\]](#)

Beschreibung: Verwenden Sie „break“, um aus einer for- oder while-Schleife auszubrechen.

C

ceil()

Modul: math

[math](#) Modul
1:math... Math
8:ceil()

Syntax: ceil(x)

Beschreibung: Gibt die kleinste ganze Zahl zurück, die größer oder gleich x ist.

[2nd](#) [\[catalog\]](#)

Beispiel:

```
>>>from math import *
>>>ceil(34.46)
35
>>>ceil(678)
678
```

[Fns...] Modul
1:math...Math
8:ceil()

Importbefehle
finden Sie unter
[2nd](#) [\[catalog\]](#).

choice(sequence)

Modul: random

[math](#) Modul
2:random...
Random
5:choice(sequence)

Syntax: choice(sequence)

Beschreibung: Liefert ein Zufallselement aus einer nicht leeren Folge.

[2nd](#) [\[catalog\]](#)

Beispiel:

```
>>>from random import *
>>>listA=[2, 4, 6, 8]
>>>choice(listA) #Ihr Ergebnis kann abweichen.
4
```

[Fns...] Modul
2:random...
Random
5:choice(sequence)

Importbefehle finden Sie
unter
[2nd](#) [\[catalog\]](#).

chr(integer)

Modul: Built-in

[\[2nd\]](#) [\[catalog\]](#)

Syntax: chr(integer)

Beschreibung: Gibt eine Zeichenfolge aus einer ganzzahligen Eingabe zurück, die das Unicode-Zeichen darstellt.

Weitere Einzelheiten finden Sie in der Python-Dokumentation.

Beispiel:

```
>>> chr(40)
'('
>>> chr(35)
'#'
```

class

Schlüsselwort

[\[2nd\]](#) [\[catalog\]](#)

Beschreibung: Verwenden Sie „class“, um eine Klasse zu erstellen. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

cls() [clear screen](#)

Modul: ti_plotlib

[\[2nd\]](#) [\[catalog\]](#)

Syntax: plt.cls() [clear screen](#)

```
[Fns...]>Modul
oder \[math\]
5:ti_plotlib...>
Setup
2:cls()
```

Beschreibung: Löscht den Shell-Bildschirm für die grafische Darstellung. Tastenkürzel werden während des Plottings nicht angezeigt.

Hinweis: plt.cls() weist ein anderes Verhalten auf als ti_system module disp_clr().

```
[Fns...]>Modul
oder \[math\]
5:ti_plotlib...>
Draw
2:cls()
```

Beispiel:

Siehe Beispielprogramm: [GRAPH](#).

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#)
oder im
ti_plotlib

color(r,g,b) 0-255**Modul:** ti_plotlib[\[2nd\]](#) [\[catalog\]](#)**Syntax:** plt.color(r,g,b) 0-255[Fns...]>Modul
oder [\[math\]](#)
5:ti_plotlib...>
Draw
1:color()**Beschreibung:** Legt die Farbe für alle folgenden Grafiken/Plots fest. Die (r,g,b)-Werte müssen mit 0-255 angegeben werden. Die angegebene Farbe wird in der Diagrammdarstellung verwendet, bis color() mit einer anderen Farbe erneut ausgeführt wird.

Die Standardfarbe ist nach Import von ti_plotlib Schwarz.

Beispiel:Siehe Beispielprogramm: [COLORLIN](#).Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#)
oder im ti_
plotlib Setup-
Menü.**complex(real,imag)****Modul:** Built-in[\[2nd\]](#) [\[catalog\]](#)**Syntax:** complex(real,imag)[Fns...]>Type>
5:complex()**Beschreibung:** Komplexer Zahlentyp.**Beispiel:**

```

>>>z = complex(2, -3)
>>>print(z)
(2-3j)
>>>z = complex(1)
>>>print(z)
(1+0j)
>>>z = complex()
>>>print(z)
0j
>>>z = complex("5-9j")
>>>print(z)
(5-9j)

```

Hinweis: "1+2j" ist die korrekte Syntax. Leerzeichen wie bei "1 + 2j" zeigen eine Ausnahme (Exception) an.

continue

Schlüsselwort

[2nd](#) [\[catalog\]](#)

Beschreibung: Verwenden Sie „continue“ in einer for- oder while-Schleife, um die aktuelle Iteration zu beenden. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

cos()

Modul: math

[\[sin\]](#) Trig

Syntax: cos(x)

4: cos()

Beschreibung: Ergibt cos von x. Das Winkelargument wird in radians (Bogenmaß) ausgegeben.

[\[2nd\]](#) [\[catalog\]](#)

Beispiel:

```
>>>from math import *
>>>cos(0)
1.0
>>>cos(pi/2)
6.123233995736767e-17
```

[Fns...] Modul
1:math... > Trig
4:cos()

Alternatives Beispiel:

```
>>>import math
>>>math.cos(0)
1.0
```

Hinweis: Python zeigt die wissenschaftliche Notation mit e oder E an. Einige math-Ergebnisse werden in einem Python anders sein als im CE-Betriebssystem.

.count()

Modul: Built-in

[\[2nd\]](#) [\[catalog\]](#)

Syntax: listname.count(item)

Beschreibung: count() ist eine Methode, die die Anzahl des Vorkommens eines Elements in einem Listen-, Tupel-, Bytes-, str-, Bytearray- oder array.array-Objekt zurückgibt.

Beispiel:

```
>>>listA = [2,4,2,6,2,8,2,10]
>>>listA.count(2)
4
```

D

def function ():

Schlüsselwort

[2nd](#) [\[catalog\]](#)

Syntax: def function(var, var,...)

Beschreibung: Definieren einer Funktion in Abhängigkeit von angegebenen Variablen. Wird typischerweise mit dem Schlüsselwort „return“ verwendet.

[Fns...]>Func
1:def function():

[Fns...]>Func
2:return

Beispiel:

```
>>> def f(a,b):  
...return a*b  
...  
...  
...  
>>> f(2,3)  
6
```

degrees()

Modul: math

[sin](#) Trig
2:degrees()

Syntax: degrees(x)

Beschreibung: Konvertiert den Winkel x von Bogenmaß nach Grad.

[2nd](#) [\[catalog\]](#)

Beispiel:

```
>>>from math import *  
>>>degrees(pi)  
180.0  
>>>degrees(pi/2)  
90.0
```

[Fns...]>Modul
1:math...>Trig
2:degrees()

del

Schlüsselwort

[2nd](#) [\[catalog\]](#)

Beschreibung: Verwenden Sie „del“, um Objekte wie Variablen, Listen usw. zu löschen. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

`disp_at(row,col,"text")`

Modul: `ti_system`

[\[2nd\]](#) [\[catalog\]](#)

Syntax: `disp_at(row,col,"text")`

[\[2nd\]](#) [\[rc1\]](#)

Beschreibung: Text anzeigen, beginnend an einer bestimmten Zeilen- und Spaltenposition im Plotting-Bereich.

`ti_system`
`7:disp_at()`

Wenn sich der Text am Ende des Programms befindet, wird hinter dem Text REPL mit dem Cursor `>>>|` angezeigt. Verwenden Sie `disp_cursor()`, um die Cursor-Anzeige zu steuern.

`[Fns...]>Modul`
oder [\[math\]](#)
`4:ti_system`
`7:disp_at()`

Argument:

<code>row</code>	<code>1 - 11, Ganzzahl</code>
<code>column</code>	<code>1 - 32, Ganzzahl</code>
<code>n</code>	
<code>"text"</code>	<code>ist ein String, der im Bildschirmbereich umgebrochen wird</code>

Importbefehle finden Sie unter [\[2nd\]](#) [\[catalog\]](#) oder im Menü des Moduls `ti_system`.

Hier gezeigte optionale Argumente für Farbe und Hintergrund: `disp_at(row,col,"text","align",color 0-15,background color 0-5)`

Beispiel:

Beispielprogramm:

```
from ti_system import *
disp_clr() #clears Shell screen
disp_at(5,6,"hello")
disp_cursor(0)
disp_wait()
```

disp_at(row,"text","align")

Modul: ti_system

[\[2nd\]](#) [\[catalog\]](#)

Syntax: disp_at(row,"text","align")

[\[2nd\]](#) [\[rc\]](#)

Beschreibung: Anzeigen von Text, wie im Plotting-Bildschirm für Zeile 1-11 angegeben. Die Zeile wird vor der Anzeige gelöscht. Bei der Verwendung in einer Schleife wird der Inhalt bei jeder Anzeige aktualisiert.

ti_system
7:disp_at()

Wenn sich der Text am Ende des Programms befindet, wird hinter dem Text REPL mit dem Cursor >>>| angezeigt. Verwenden Sie disp_cursor() zur Steuerung der Cursor-Anzeige in Ihrem Programm vor disp_at().

[Fns...]>Modul
oder [\[math\]](#)
4:ti_system
7:disp_at()

Argument:

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#)
oder im Menü
des Moduls
ti_system.

row	1 - 11, Ganzzahl
"text"	ist ein String, der im Bildschirmbereich umgebrochen wird
"align"	"left" (Standard) "center" "right"

Hier gezeigtes optionales Argument: disp_at
(row,col,"text","align",color 0-15, 0-15)

Beispiel:

Beispielprogramm:

```
from ti_system import *
disp_clr() #clears Shell screen
disp_at(5,"hello","left")
disp_cursor(0)
disp_wait()
```

disp_clr() clear text screen

Modul: ti_system

[\[2nd\]](#) [\[catalog\]](#)

Syntax: disp_clr() [clear text screen](#)

[\[2nd\]](#) [\[rc1\]](#)

Beschreibung: Löscht den Bildschirm in der Shell-Umgebung. „Row 0-11, integer“ kann als optionales Argument verwendet werden, um eine Anzeigzeile der Shell-Umgebung zu löschen.

ti_system
8:disp_clr()

Beispiel:

[Fns...]>Modul
oder [\[math\]](#)
4:ti_system
8:disp_clr()

Beispielprogramm:

```
from ti_system import *
disp_clr() #clears Shell screen
disp_at(5, "hello", "left")
disp_cursor(0)
disp_wait()
```

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#)
oder im Menü
des Moduls
ti_system.

`disp_cursor()` 0=off 1=on

Modul: `ti_system`

[\[2nd\]](#) [\[catalog\]](#)

Syntax: `disp_cursor()` 0=off 1=on

[\[2nd\]](#) [\[rc\]](#)

Beschreibung: Steuert die Anzeige des Cursors in der Shell, wenn ein Programm ausgeführt wird.

`ti_system`
0:`disp_cursor()`

Argument:

[Fns...]>Modul oder

0 = aus

[math](#)

nicht 0 = ein

4:`ti_system`

0:`disp_cursor()`

Beispiel:

Beispielprogramm:

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#) oder
im Menü des
Moduls
`ti_system`.

```
from ti_system import *
disp_clr() #clears Shell screen
disp_at(5, "hello", "left")
disp_cursor(0)
disp_wait()
```

disp_wait() [clear]

Modul: ti_system

[2nd] [catalog]

Syntax: disp_wait() [clear]

[2nd] [rel]

Beschreibung: Stoppt die Ausführung des Programms an diesem Punkt und zeigt den Bildschirminhalt an, bis [clear] gedrückt und der Bildschirm gelöscht wird.

ti_system
9:disp_wait()

Beispiel:

[Fns...]>Modul
oder **[math]**

Beispielprogramm:

4:ti_system
9:disp_wait()

```
from ti_system import *  
disp_clr() #clears Shell screen  
disp_at(5, "hello", "left")  
disp_cursor(0)  
disp_wait()
```

Importbefehle
finden Sie
unter **[2nd]**
[catalog] oder
im Menü des
Moduls
ti_system.

E

e

Modul: math

 [e] (über
)

Syntax: math.e oder e bei importiertem math-Modul

Beschreibung: Die Konstante e wird wie unten dargestellt angezeigt.

[Fns...] > Modul
1:math...
> Const 1:e

Beispiel:

```
>>>from math import *  
>>>e  
2.718281828459045
```

Alternatives Beispiel:

```
>>>import math  
>>>math.e  
2.718281828459045
```

elif :

Schlüsselwort

 [catalog]

Einzelheiten finden Sie unter if..elif..else...

[Fns...] > Ctl

1:if..

2:if..else..

3:if..elif..else

9:elif :

0:else:

else:

Schlüsselwort

[\[2nd\]](#) [\[catalog\]](#)

Einzelheiten finden Sie unter `if..elif..else...`

`[Fns...] > Ctl`

`1:if..`

`2:if..else..`

`3:if..elif..else`

`9:elif :`

`0:else:`

escape()

Modul: `ti_system`

[\[2nd\]](#) [\[catalog\]](#)

Syntax: `escape()`

Als

Programmzeile:

Beschreibung: `escape()` gibt True oder False zurück.

Der Anfangswert ist False.

Wenn die Taste `[clear]` auf dem CE gedrückt wird, wird der Wert auf True gesetzt.

Beim Ausführen des Programms wird der Wert auf False zurückgesetzt.

[\[2nd\]](#) [\[rc1\]](#)

`ti_system`

`5:while not`

`escape():`

`6:if escape`

`():``break`

Anwendungsbeispiel:

`[Fns...]>Modul`

oder [\[math\]](#)

`4:ti-system`

`5:while not`

`escape():`

`6:if escape`

`():``break`

`while not escape():`

In einer `while`-Schleife eines Programms, wobei das Programm anbietet, die Schleife zu beenden, aber das Skript weiterlaufen zu lassen.

`if escape():break`

Importbefehle

finden Sie unter

[\[2nde\]](#) [\[catalog\]](#)

oder im Menü

des Moduls

`ti_system`.

Kann in einem Programm für die Fehlersuche verwendet werden, um Variablen mit Shell `[vars]` zu untersuchen, nachdem das Programm ausgeführt und diese Pause verwendet wurde.

eval()

Modul: Built-in

[2nd](#) [\[catalog\]](#)

Syntax: eval(x)

[Fns...] I/O
3:eval()

Beschreibung: Gibt die Auswertung des Ausdrucks x zurück.

Beispiel:

```
>>>a=7
>>>eval("a+9")
16
>>>eval('a+10')
17
```

except **exception**:

Schlüsselwort

[2nd](#) [\[catalog\]](#)

Beschreibung: Verwenden Sie „except“ in einem try..except-Codeblock. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

exp() (e hoch x)

Modul: math

[2nd](#) [e^x] (über [in](#))

Syntax: exp(x)

Beschreibung: Gibt e**x zurück.

[2nd](#) [catalog]

Beispiel:

```
>>>from math import *
>>>exp(1)
2.718281828459046
```

```
[Fns...] > Modul
1:math...
4:exp()
```

Alternatives Beispiel: [Tools] > 6:New Shell

```
>>>import math
>>>math.exp(1)
2.718281828459046
```

Importbefehle
finden Sie unter
[2nd](#) [catalog][format].

.extend()

Modul: Built-in

[2nd](#) [catalog]

Syntax: listname.extend(newlist)

Beschreibung: Die Methode extend() ist eine Methode zum Erweitern von „newlist“ bis zum Ende einer Liste.

Beispiel:

```
>>>listA = [2,4,6,8]
>>>listA.extend([10,12])
>>>print(listA)
[2,4,6,8,10,12]
```

F

fabs()

Modul: math

[\[2nd\]](#) [\[catalog\]](#)

Syntax: fabs(x)

Beschreibung: Gibt den Absolutwert von x zurück.

[Fns...] > Modul

1:math...

2:fabs()

Beispiel:

```
>>>from math import *
>>>fabs(35-65.8)
30.8
```

Importbefehle
finden Sie unter

[\[2nd\]](#) [\[catalog\]](#)[\[format\]](#).

Siehe auch die
integrierte Funktion
abs().

False

Schlüsselwort

[\[2nd\]](#) [\[test\]](#) (über
[\[math\]](#))

Beschreibung: Gibt False zurück, wenn die ausgeführte Anweisung False ist. „False“ steht für einen falschen Wert bei Objekten des Typs „bool“.

[\[2nd\]](#) [\[catalog\]](#)

Beispiel:

```
>>>64<=32
False
```

[Fns...] > Ops

B:False

[a A #]

finally:

Schlüsselwort

[2nd](#) [\[catalog\]](#)

Beschreibung: Verwenden Sie "finally" in einem try..except..finally-Codeblock. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

float()

Modul: Built-in

[2nd](#) [\[catalog\]](#)

Syntax: float(x)

Beschreibung: Gibt x als Fließkommazahl zurück.

[Fns...] > Type
2:float()

Beispiel:

```
>>>float(35)
35.0
>>>float("1234")
1234.0
```

floor()

Modul: math

[math](#) Modul
1:math
9:floor()

Syntax: floor(x)

Beschreibung: Gibt die größte ganze Zahl zurück, die kleiner oder gleich x ist.

[2nd](#) [\[catalog\]](#)

Beispiel:

```
>>>from math import *
>>>floor(36.87)
36
>>>floor(-36.87)
-37
>>>floor(254)
254
```

[Fns...] > Modul
1:math
9:floor()

Importbefehle
finden Sie unter
[2nd](#) [\[catalog\]](#).

fmod(x,y)

Modul: math

[math](#) Modul

Syntax: fmod(x,y)

1:math
7:fmod()

Beschreibung: Weitere Einzelheiten finden Sie in der Python-Dokumentation. Wird bevorzugt verwendet, wenn x und y Fließkommazahlen sind.

[2nd](#) [\[catalog\]](#)

Gibt möglicherweise nicht das gleiche Ergebnis zurück wie x%y.

Beispiel:

[Fns...] > Modul
1:math...
7:fmod()

```
>>>from math import *  
>>>fmod(50.0,8.0)  
2.0  
>>>fmod(-50.0,8.0)  
-2.0  
>>>-50.0 - (-6.0)*8.0 #Validierung aus der Beschreibung  
-2.0
```

Importbefehle
finden Sie unter
[2nd](#) [\[catalog\]](#).

Siehe auch: x%y.

for i in list:

Schlüsselwort

[Fns...] Ctl

Syntax: for i in list:

7:for i in list:

Beschreibung: Zum Iterieren über Listenelemente.

[2nd](#) [\[catalog\]](#)

Beispiel:

```
>>> for i in [2,4,6]:  
... print(i)  
...  
...  
...  
2  
4  
6
```

for i in range(size):

Schlüsselwort

[Fns...] Ctl

Syntax: for i in range(size)

4:for i in range
(size):

Beschreibung: Zum Iterieren über einen Bereich.

Beispiel:

[2nd](#) [catalog]

```
>>> for i in range(3):  
... print(i)  
...  
...  
...  
0  
1  
2
```

for i in range(start,stop):

Schlüsselwort

[Fns...] Ctl

Syntax: for i in range(start,stop)

5:for i in range
(start,stop):

Beschreibung: Zum Iterieren über einen Bereich.

Beispiel:

[2nd](#) [catalog]

```
>>> for i in range(1,4):  
... print(i)  
...  
...  
...  
1  
2  
3
```

for i in range(start,stop,step):

Schlüsselwort

[Fns...] Ctl

Syntax: for i in range(start,stop,step)

6:for i in range
(start,stop,step):

Beschreibung: Zum Iterieren über einen Bereich.

Beispiel:

[2nd](#) [catalog]

```
>>> for i in range(1,8,2):  
... print(i)  
...  
...  
...  
1  
3  
4  
7
```

str.format() string format

Modul: Built-in

[2nd](#) [catalog]

Syntax:str.format()

Beschreibung: Formatiert die angegebene Zeichenkette. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

Beispiel:

```
>>> print("{+f}".format(12.34))  
+12.340000
```

frexp()

Modul: math

[math](#) Modul

Syntax: frexp(x)

1:math
A:frexp()

Beschreibung: Gibt ein Paar (y,n) zurück, wobei $x == y * 2^{**}n$. y ist eine Fließkommazahl, wobei $0.5 < \text{abs}(y) < 1$; und n ist eine Ganzzahl.

[2nd](#) [\[catalog\]](#)

Beispiel:

```
>>>from math import *
>>>frexp(2000.0)
(0.9765625, 11)
>>>0.9765625 * 2**11 #Beschreibung validieren
2000.0
```

[Fns...] > Modul
1:math
A:frexp()

Importbefehle
finden Sie unter
[2nd](#) [\[catalog\]](#).

from PROGRAM import *

Schlüsselwort

Shell [Tools]

Syntax: from PROGRAM import *

A:from
PROGRAM
import *

Beschreibung: Zum Importieren eines Programms. Importiert die öffentlichen Attribute eines Python-Moduls in den aktuellen Namensraum.

[2nd](#) [\[catalog\]](#)

from math import *

Schlüsselwort

Syntax: from math import *

Beschreibung: Zum Import aller Funktionen und Konstanten aus dem math module.

`[math]` Modul
1:math...
1:from math
import *

[Fns..] > Modul
1:math...
1:from math
import *

`[2nd]` [catalog]

from random import *

Schlüsselwort

Syntax: from random import *

Beschreibung: Zum Import aller Funktionen aus dem random module.

`[math]` Modul
2:random...
1:from random
import *

[Fns..] > Modul
2:random...
1:from random
import *

`[2nd]` [catalog]

from time import *

Schlüsselwort

[2nd] [catalog]

Syntax: from time import *

[math] Modul

Beschreibung: Zum Import aller Methoden aus dem time-Modul.

3:time...

1:from time import

*

Beispiel:

[Fns...]>Modul

Siehe Beispielprogramm: [DASH1](#).

3:time...

1:from time import

*

from ti_system import *

Schlüsselwort

[2nd] [catalog]

Syntax: from ti_system import *

[math] Modul

Beschreibung: Zum Import aller Methoden aus dem ti_system-Modul.

4:ti_system...

1:from system

import *

Beispiel:

[Fns...]>Modul

Siehe Beispielprogramm: [REGEQ1](#).

4:ti_system...

1:from system

import *

```
from ti_hub import *
```

Schlüsselwort

[2nd] [catalog]

Syntax: from ti_hub import *

Beschreibung: Zum Import aller Methoden aus dem ti_hub module. Verwenden Sie für einzelne Ein- und Ausgabegeräte die dynamische Modulfunktionalität, indem Sie das Gerät im Editor im Menü [Fns...]>Modul>ti_hub>Import auswählen.

Siehe [ti_hub module – Import zum Editor hinzufügen und ti_hub-Sensormodul zum Modul-Menü hinzufügen.](#)

Beispiel:

Siehe Beispielprogramm: [DASH1](#).

G

global

Schlüsselwort

[2nd](#) [\[catalog\]](#)

Beschreibung: Verwenden Sie „global“, um globale Variablen innerhalb einer Funktion zu erstellen.

Weitere Einzelheiten finden Sie in der CircuitPython-Dokumentation.

grid(xscl,yscl,"style")

Modul: `ti_plotlib`

[2nd](#) [\[catalog\]](#)

Syntax: `plt.grid(xscl,yscl,"style")`

[Fns...]>Modul
oder [math](#)
5:ti_plotlib...>
Setup
3:grid()

Beschreibung: Zeigt ein Gitter mit angegebenem Maßstab für die x- und y-Achse an. Hinweis: Das Plotting wird durchgeführt, wenn `plt.show_plot()` ausgeführt wird.

Die Einstellung der Gitterfarbe ist das optionale Argument von `(r,g,b)` unter Verwendung der Werte [0-255](#) mit dem Standardwert Grau (192,192,192).

Standardwert für `xscl` oder `yscl` = 1.0.

`"style"` = "dot" (Standard), "dash", "solid" oder "point"

Importbefehle
finden Sie unter
[2nd](#) [\[catalog\]](#)
oder im Menü
ti_plotlib Setup.

Beispiel:

Siehe Beispielprogramme: [COLORLIN](#) oder [GRAPH](#).

`grid(xscl,yscl,"style",(r,g,b))`

Modul: `ti_plotlib`

[\[2nd\]](#) [\[catalog\]](#)

Syntax: `plt.grid(xscl,yscl,"style",(r,g,b))`

[Fns...]>Modul
oder [\[math\]](#)
5:ti_plotlib...>
Setup
3:grid()

Beschreibung: Zeigt ein Gitter mit angegebenem Maßstab für die x- und y-Achse an. Hinweis: Das Plotting wird durchgeführt, wenn `plt.show_plot()` ausgeführt wird.

Die Einstellung der Gitterfarbe ist das optionale Argument von `(r,g,b)` unter Verwendung der Werte [0-255](#) mit dem Standardwert Grau (192,192,192).

Standardwert für `xscl` oder `yscl` = 1.0.

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#)
oder im Menü
ti_plotlib Setup.

`"style"` = "dot" (Standard), "dash", "solid" oder "point".

Wenn die Werte `xscl` oder `yscl` kleiner als 1/50 der Differenz zwischen `xmax-xmin` oder `ymax-ymin` betragen, dann wird eine Ausnahme 'Invalid grid scale value' ausgegeben.

Beispiel:

Siehe Beispielprogramm: [GRAPH](#).

H

hex([integer](#))

Modul: Built-in

[2nd](#) [catalog](#)

Syntax: hex([integer](#))

Beschreibung: Zeigt das hexadezimale Format des Arguments „integer“ an. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

Beispiel:

```
>>> hex(16)
'0x10'
>>> hex(16**2)
'0x100'
```

"if :"Einzelheiten finden Sie unter `if..elif..else...`[\[2nd\]](#) [\[catalog\]](#)

[Fns...] > Ctl

1:if..

2:if..else..

3:if..elif..else

9:elif :

0:else:

if..elif..else..

Schlüsselwort

2nd [catalog]

Syntax: ••Graue eingerückte Identifikatoren werden in der Python-App aus Gründen der Benutzerfreundlichkeit automatisch bereitgestellt.

[Fns...] > Ctl

if :

1:if..

••

2:if..else..

elif :

3:if..elif..else

••

9:elif :

else:

0:else:

Beschreibung: if..elif..else ist eine bedingte Anweisung. Der Editor fügt Einzüge automatisch als graue Punkte ein, um Sie bei der korrekten Programmierung der Einzüge zu unterstützen.

Beispiel: Erstellen und starten Sie ein Programm, z. B. S01, aus dem Editor heraus.

```
def f(a):
    ••if a>0:
        •••print(a)
    ••elif a==0:
        •••print("zero")
    ••else:
        •••a=-a
    •••print(a)
```

Shell-Interaktion

```
>>> # Shell Reinitialized
>>> # Running S01
>>>from S01 import *#fügt automatisch ein
>>>f(5)
5
>>>f(0)
Null
>>>f(-5)
5
```

if..else..

Schlüsselwort

[2nd] [catalog]

Einzelheiten finden Sie unter if..elif..else...

[Fns...] > Ctl

1:if..

2:if..else..

3:if..elif..else

9:elif :

0:else:

.imag

Modul: Built-in

[2nd] [catalog]

Syntax: var.imag

Beschreibung: Bestimmt den Imaginärteil einer angegebenen Variablen eines komplexen Zahlentyps.

Beispiel:

```
>>>a=complex(4,5)
>>>a.real
4
>>>a.imag
5
```

import math

Schlüsselwort

Syntax: import math

[2nd] [catalog]

Beschreibung: Über diesen Befehl wird auf das math module zugegriffen. Diese Anweisung importiert die öffentlichen Attribute des Moduls „math“ innerhalb seines eigenen Namensraums.

import random

Schlüsselwort

Syntax: import random

[2nd] [catalog]

Beschreibung: Über diesen Befehl wird auf das random module zugegriffen. Diese Anweisung importiert die öffentlichen Attribute des Moduls „random“ innerhalb seines eigenen Namensraums.

import ti_hub

Schlüsselwort

[2nd] [catalog]

Syntax: import ti_hub

Beschreibung: Über diesen Befehl wird auf das ti_hub module zugegriffen. Diese Anweisung importiert die öffentlichen Attribute von ti_hub module innerhalb seines eigenen Namensraums.

Verwenden Sie für einzelne Ein- und Ausgabegeräte die dynamische Modulfunktionalität, indem Sie das Gerät im Editor im Menü [Fns...]>Modul>ti_hub>Import auswählen.

Siehe: [\[Fns...\] > Modul: ti_hub module](#).

import time

Schlüsselwort

[2nd] [catalog]

Syntax: import time

Beschreibung: Über diesen Befehl wird auf das time module zugegriffen. Diese Anweisung importiert die öffentlichen Attribute des Moduls „time“ innerhalb seines eigenen Namensraums.

Siehe: [\[Fns...\] > Modul: time and ti_system modules](#).

import ti_plotlib as plt

Schlüsselwort

2nd [catalog]

Syntax: import ti_plotlib as plt

math Modul
5:ti_plotlib...
1:import ti_plotlib
as plt

Beschreibung: Über diesen Befehl wird auf das ti_plotlib module zugegriffen. Diese Anweisung importiert die öffentlichen Attribute von ti_plotlib module innerhalb seines eigenen Namensraums. Attribute von ti_plotlib module müssen als plt.attribute eingegeben werden.

[Fns...]>Modul
5:ti_plotlib...
1:import ti_plotlib
as plt

Beispiel:

Siehe Beispielprogramm: [COLORLIN](#).

import ti_rover as rv

Schlüsselwort

2nd [catalog]

Syntax: import ti_rover as rv

math Modul
7:ti_rover...
1:import ti_rover
as rv

Beschreibung: Über diesen Befehl wird auf das ti_rover module zugegriffen. Diese Anweisung importiert die öffentlichen Attribute von ti_rover module innerhalb seines eigenen Namensraums. Attribute von ti_rover module müssen als rv.attribute eingegeben werden.

[Fns...]>Modul
7:ti_rover...
1:import ti_rover
as rv

Beispiel:

Siehe Beispielprogramm: [ROVER](#).

import ti_system

Schlüsselwort

2nd [catalog]

Syntax: import ti_system

Beschreibung: Über diesen Befehl wird auf das ti_system module zugegriffen. Diese Anweisung importiert die öffentlichen Attribute von ti_system module innerhalb seines eigenen Namensraums.

Beispiel:

Siehe Beispielprogramm: [REGEQ1](#).

in

Schlüsselwort

2nd [catalog]

Beschreibung: Zur Überprüfung, ob ein Wert in einer Folge enthalten ist oder um zum Iterieren einer Folge in einer for-Schleife.

.index(x)

Modul: Built-in

2nd [catalog]

Syntax: var.index(x)

Beschreibung: Gibt den Index oder die Position eines Elements einer Liste zurück. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

Beispiel:

```
>>> a=[12,35,45]
>>> print(a.index(12))
0
>>> print(a.index(35))
1
>>> print(a.index(45))
2
```

input()

Modul: Built-in

2nd [catalog]

input()

Syntax: input()

Beschreibung: Eingabeaufforderung

[Fns...] I/O
2:input()

Beispiel:

```
>>>input("Name? ")
Name? Me
'Me'
```

Alternatives Beispiel:

```
CreateProgram A
len=float(input("len: "))
print(len)
```

```
RunProgram A
>>> # Shell Reinitialized
>>> # Running A
>>>from A import *
len: 15(enter15)
15.0 (outputfloat 15.0)
```

.insert(index,x)

Modul: Built-in

[2nd](#) [\[list\]](#) List
8:.insert(index,x)

Syntax: listname.insert(index,x)

Beschreibung: method insert() fügt innerhalb einer Folge ein Element x nach dem Index ein.

[2nd](#) [\[catalog\]](#)

Beispiel:

```
>>>listA = [2,4,6,8]
>>>listA.insert(3,15)
>>>print(listA)
[2, 4, 6, 15, 8]
```

[\[Fns...\]](#) > List
8:.insert(index,x)

int()

Modul: Built-in

[2nd](#) [\[catalog\]](#)

Syntax: int(x)

Beschreibung: Gibt x als ganzzahliges Objekt zurück.

[\[Fns...\]](#) > Type
1:int()

Beispiel:

```
>>>int(34.67)
34
>>>int(1234.56)
1234
```

is

Schlüsselwort

[2nd](#) [\[catalog\]](#)

Beschreibung: Verwenden Sie „is“, um zu testen, ob zwei Objekte das gleiche Objekt sind.

labels("xlabel","ylabel",x,y)**Modul:** ti_plotlib[\[2nd\]](#) [\[catalog\]](#)**Syntax:** plt.labels("xlabel","ylabel",x,y)[Fns...]>Modul
oder [\[math\]](#)
5:ti_plotlib...>
Setup
7:labels()**Beschreibung:** Zeigt die Beschriftungen „xlabel“ und „ylabel“ an den Diagrammachsen in Höhe der Zeilenpositionen x und y an. Passen Sie sie ggf. an die Diagrammdarstellung an.

„xlabel“ wird auf der angegebenen Zeile x (Standardzeile: 12) rechtsbündig positioniert.

„ylabel“ wird auf der angegebenen Zeile y (Standardzeile: 2) linksbündig positioniert.

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#)
oder im Menü
ti_plotlib Setup.**Hinweis:** plt.labels("|","",12,2) wird mit den Standardwerten für die x- und y-Zeile (12,2) eingefügt; dies kann anschließend für Ihr Programm modifiziert werden.**Beispiel:**Siehe Beispielprogramm: [GRAPH](#).**lambda****Schlüsselwort**[\[2nd\]](#) [\[catalog\]](#)**Syntax:** lambda arguments : expression**Beschreibung:** Verwenden Sie „lambda“, um eine anonyme Funktion zu definieren. Einzelheiten finden Sie in der Python-Dokumentation.

len()

Modul: Built-in

[2nd](#) [list] (über
[stat](#)) List
3:len()

Syntax: len(sequence)

Beschreibung: Gibt die Anzahl der Elemente im Argument zurück. Das Argument kann eine Folge oder eine Sammlung sein.

[2nd](#) [catalog]

Weitere Einzelheiten finden Sie in der Python-Dokumentation.

Beispiel:

[Fns...] > List
3:len()

```
>>>mylist=[2,4,6,8,10]
>>>len(mylist)
5
```

line(x1,y1,x2,y2,"mode")

Modul: tiplotlib

[2nd](#) [catalog]

Syntax: plt.line(x1,y1,x2,y2,"mode")

[Fns...]>Modul oder
[math](#)
5:tiplotlib...>
Draw
7:line or vector

Beschreibung: Zeigt eine Strecke von (x1,y1) bis (x2,y2) an.

Größe und Stil werden mittels pen() und color() vor line() festgelegt.

Argumente:

x1,y1, x2,y2 sind echte Fließkommazahlen.

"mode": Bei Standard "" wird keine Pfeilspitze gezeichnet.

Bei "arrow" wird eine Vektorpfeilspitze bei (x2,y2) gezeichnet.

Importbefehle
finden Sie unter
[2nd](#) [catalog] oder
im Menü
tiplotlib Setup.

Beispiel:

Siehe Beispielprogramm: [COLORLIN](#).

lin_reg(xlist,ylist,"disp",row)

Modul: ti_plotlib

[\[2nd\]](#) [\[catalog\]](#)

Syntax: plt.lin_reg(xlist,ylist,"disp",row)

[Fns...]>Modul
oder [math](#)
5:ti_plotlib...>
Draw
8:lin_reg()

Beschreibung: Berechnet und zeichnet das lineare Regressionsmodell $ax+b$ von xlist,ylist. Diese Methode muss nach der scatter-Methode ausgeführt werden. Die Standardanzeige der Gleichung ist "center" in Zeile 11.

Argument:

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#)
oder im Menü
ti_plotlib Setup.

"disp"	"left"
	"center"
	"right"
row	1 - 12

plt.a (Steigung) und plt.b (Achsenabschnitt) werden gespeichert, wenn lin_reg ausgeführt wird.

Beispiel:

Siehe Beispielprogramm: [LINREGR](#).

list(sequence)

Modul: Built-in

[2nd](#) [\[list\]](#) (über
[stat](#)) List
2:list(sequence)

Syntax: list(sequence)

Beschreibung: Veränderbare Folge von Elementen desselben Typs.

[2nd](#) [\[catalog\]](#)

list() konvertiert die Argumente in den Typ „list“. Wie bei vielen anderen Folgen müssen die Elemente einer Liste nicht vom selben Typ sein.

[Fns...] > List
2:list(sequence)

Beispiel:

```
>>>mylist=[2,4,6,8]
>>>print(mylist)
[2,4,6,8]
```

Beispiel:

```
>>>mylist=[2,4,6,8]
>>>print(mylist)
[2,4,6,8]
>>> list({1,2,"c", 7})
[7, 1, 2, 'c']
>>> list("foobar")
['f', 'o', 'o', 'b', 'a', 'r']
```

log(x,base)

Modul: math

[\[2nd\]](#) [\[log\]](#) for log
(x,10)

Syntax: log(x,base)

Beschreibung: log(x) ohne Basis gibt den natürlichen Logarithmus x zurück.

[\[2nd\]](#) [\[ln\]](#) for log
(x) (natürlicher
Logarithmus)

Beispiel:

```
>>>from math import *
>>>log(e)
1.0
>>>log(100,10)
2.0
>>>log(32,2)
5.0
```

[\[math\]](#) Modul
1:math...
6:log(x,base)

[\[2nd\]](#) [\[catalog\]](#)

[Fns...] > Modul
1:math...
6:log(x,base)

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#).

math.function**Modul:** math[2nd](#) [\[catalog\]](#)**Syntax:** math.function**Beschreibung:** Ist nach dem Befehl „import math“ zu benutzen, um eine Funktion im math-Modul zu verwenden.**Beispiel:**

```
>>>import math
>>>math.cos(0)
1.0
```

max()**Modul:** Built-in[2nd](#) [\[list\]](#) (über [stat](#)) List
4:max()**Syntax:** max(sequence)**Beschreibung:** Gibt den Maximalwert der Folge zurück. Weitere Informationen zu max() finden Sie in der Python-Dokumentation.[2nd](#) [\[catalog\]](#)**Beispiel:**

```
>>>listA=[15,2,30,12,8]
>>>max(listA)
30
```

[Fns...] > List
4:max()**min()****Modul:** Built-in[2nd](#) [\[list\]](#) (über [stat](#)) List
5:min()**Syntax:** min(sequence)**Beschreibung:** Gibt den Minimalwert der Folge zurück. Weitere Informationen zu min() finden Sie in der Python-Dokumentation.[2nd](#) [\[catalog\]](#)**Beispiel:**

```
>>>listA=[15,2,30,12,8]
>>>min(listA)
2
```

[Fns...] > List
5:min()

monotonic() elapsed time

Modul: time

[\[2nd\]](#) [\[catalog\]](#)

Syntax: monotonic() elapsed time

Beschreibung: Gibt einen Zeitwert vom Zeitpunkt der Ausführung zurück. Verwenden Sie den Rückgabewert zum Vergleich mit anderen Werten aus monotonic().

[Fns...]>Modul
oder [\[math\]](#)
3:time
3:monotonic()

Beispiel:

Beispielprogramm:

```
from time import *  
a=monotonic()  
sleep(15)  
b=monotonic()  
print(b-a)
```

Importbefehle
finden Sie
unter [\[2nd\]](#)
[\[catalog\]](#) oder
im Menü des
Moduls time.

Run the program EXAMPLE until execution stops.
>>>15.0

N

None

Schlüsselwort

[2nd](#) [\[catalog\]](#)

Beschreibung: „None“ stellt das Fehlen eines Wertes dar.

[\[a A #\]](#)

Beispiel:

```
>>> def f(x):
...x
...
...
...
>>> print(f(2))
None
```

nonlocal

Schlüsselwort

[2nd](#) [\[catalog\]](#)

Syntax: nonlocal

Beschreibung: Verwenden Sie „nonlocal“, um festzulegen, dass eine Variable nicht lokal ist. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

not

Schlüsselwort

[2nd](#) [\[test\]](#) Ops

Syntax: not x

0: not

Beschreibung: Wertet als True aus, wenn x False ist, anderenfalls als False. Wird mit Leerzeichen vor und hinter dem Schlüsselwort „not“ eingefügt. Kann nach Bedarf bearbeitet werden.

[\[Fns...\]](#) > Ops 0: not

Beispiel:

[2nd](#) [\[catalog\]](#)

```
>>> not 2<5 #Bearbeiten Sie das Leerzeichen vor „not“
False
>>>3<8 and not 2<5
False
```

[\[a A #\]](#)

O

oct(integer)

Modul: Built-in

[\[2nd\]](#) [\[catalog\]](#)

Syntax: oct(integer)

Beschreibung: Gibt die oktale Darstellung der Ganzzahl zurück. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

Beispiel:

```
>>> oct(8)
'0o10'
>>> oct(64)
'0o100'
```

or

Schlüsselwort

[\[2nd\]](#) [\[test\]](#) Ops 9:or

Syntax: x or y

[\[Fns...\]](#) > Ops 9:or

Beschreibung: Kann True oder False zurückgeben. Gibt x zurück, wenn x als True ausgewertet wird, anderenfalls wird y ausgegeben. Wird mit Leerzeichen vor und hinter dem „or“ eingefügt. Kann nach Bedarf bearbeitet werden.

[\[2nd\]](#) [\[catalog\]](#)

Beispiel:

[\[a A #\]](#)

```
>>>2<5 or 5<10
True
>>>2<5 or 15<10
True
>>>12<5 or 15<10
False
>>> 3 or {}
3
>>> [] or {2}
{2}
```

ord("character")

Modul: Built-in

[\[2nd\]](#) [\[catalog\]](#)

Syntax: ord("character")

Beschreibung: Gibt den Unicode-Wert des Zeichens zurück. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

Beispiel:

```
>>> ord("#")
35
>>> ord("/")
47
```

pass**Schlüsselwort**[\[2nd\]](#) [\[catalog\]](#)

Beschreibung: Verwenden Sie „pass“ in einer leeren Funktions- oder Klassendefinition als Platzhalter für zukünftigen Code, während Sie Ihr Programm aufbauen. Leere Definitionen führen bei der Programmausführung nicht zu einem Fehler.

pen("size", "style")**Modul:** `ti_plotlib`[\[2nd\]](#) [\[catalog\]](#)**Syntax:** `plt.pen("size", "style")`

[Fns...]>Modul oder

[\[math\]](#)

Beschreibung: Legt das Aussehen aller folgenden Zeilen fest, bis ein neues pen() ausgeführt wird.

5:ti_plotlib...> Draw

9:pen()

Argument:

Standardmäßig ist pen() "thin" und "solid".

Importbefehle

finden Sie unter

[\[2nd\]](#) [\[catalog\]](#) oder

im Menü

ti_plotlib Setup.

"size"	"thin"
	"medium"
	"thick"
"style"	"solid"
	"dot"
	"dash"

Beispiel:Siehe Beispielprogramme: [COLORLIN](#) oder [GRAPH](#).

pi

Modul: math

 [π] (über
 sin)

Syntax: math.pi oder pi bei importiertem math-Modul.

Beschreibung: Die Konstante pi wird wie unten dargestellt angezeigt.

[Fns...] > Modul
1:math... > Const
2:pi

Beispiel:

```
>>>from math import *  
>>>pi  
3.141592653589793
```

Alternatives Beispiel:

```
>>>import math  
>>>math.pi  
3.141592653589793
```

plot(xlist,ylist,"mark")

Modul: ti_plotlib

[2nd] [catalog]

Syntax: plt.plot(xlist,ylist,"mark")

[Fns...]>Modul

oder **[math]**

5:ti_plotlib...>

Draw

5:Connected Plot

with Lists

Beschreibung: Es wird ein Liniendiagramm mit geordneten Paaren aus der angegebenen xlist und ylist angezeigt. Liniestärke und -größe werden unter Verwendung von plt.pen() eingestellt.

xlist und ylist müssen echte Fließkommazahlen sein und Listen müssen die gleiche Dimension haben.

Argument:

Importbefehle

finden Sie unter

[2nd] [catalog]

oder im Menü

ti_plotlib Setup.

„mark“ ist das Markierungszeichen, wie nachstehend erläutert:

-
- o Gefüllter Punkt
 (Standard)
 - + Kreuz
 - x x
 - . Pixel
-

Beispiel:

Siehe Beispielprogramm: [LINREGR](#).

plot(x,y,"mark")

Modul: ti_plotlib

[2nd] [catalog]

Syntax: plt.plot(x,y,"mark")

[Fns...]>Modul

oder [math]

Beschreibung: Ein Punktdiagramm (x,y) zeigt den angegebenen x- und y-Wert an.

5:ti_plotlib...>

Draw

xlist und ylist müssen echte Fließkommazahlen sein und Listen müssen die gleiche Dimension haben.

6:plot a Point

Argument:

Importbefehle

finden Sie

unter [2nd]

[catalog] oder

im Menü

ti_plotlib

Setup.

„mark“ ist das Markierungszeichen, wie nachstehend erläutert:

-
- o Gefüllter Punkt
 (Standard)
 - + Kreuz
 - x x
 - . Pixel
-

Beispiel:

Siehe Beispielprogramm: [LINREGR](#).

pow(x,y)

Modul: math

[math](#) Modul

Syntax: pow(x,y)

1:math
5:pow(x,y)

Beschreibung: Ergibt x hoch y. Konvertiert x und y in Fließkommazahlen. Weitere Informationen finden Sie in der Python-Dokumentation.

[2nd](#) [catalog]

Verwenden Sie zur Berechnung exakter ganzzahliger Potenzen die integrierte pow(x,y)-Funktion oder **.

[Fns...] > Modul
1:math
5:pow(x,y)

Beispiel:

```
>>>from math import *  
>>>pow(2,3)  
>>>8.0
```

Beispiel unter Verwendung von: Built-in:

Importbefehle
finden Sie unter
[2nd](#) [catalog].

[Tools] > 6:New Shell

```
>>>pow(2,3)  
8  
>>>2**3  
8
```

print()

Modul: Built-in

[2nd](#) [catalog]

Syntax: print(argument)

Beschreibung: Zeigt das Argument als Zeichenfolge an.

[Fns...] > I/O
1:print()

Beispiel:

```
>>>x=57.4  
>>>print("my number is =", x)  
my number is= 57.4
```

R

radians() [degree](#) ▶ [radians](#)

Modul: math

[sin](#) Trig
1:radians()

Syntax: radians(x)

Beschreibung: Konvertiert den Winkel x von degrees nach radians.

[2nd](#) [catalog]

Beispiel:

```
>>>from math import *  
>>>radians(180.0)  
3.141592653589793  
>>>radians(90.0)  
1.570796326794897
```

[Fns...] > Modul
1:math... > Trig
1:radians()

raise

Schlüsselwort

[2nd](#) [catalog]

Syntax: raise exception

Beschreibung: Verwenden Sie „raise“, um eine bestimmte Ausnahme auszulösen und Ihr Programm zu stoppen.

randint(min,max)

Modul: random

[math](#) Modul

Syntax: randint(min,max)

2:random
4:randint
(min,max)

Beschreibung: Ergibt eine ganzzahlige Zufallszahl zwischen min und max.

Beispiel:

```
>>>from random import *  
>>>randint(10,20)  
>>>15
```

[Fns...] > Modul
2:random...
4:randint
(min,max)

Alternatives Beispiel:

```
>>>import random  
>>>random.randint(200,450)  
306
```

[2nd](#) [catalog]

Die Ergebnisse variieren aufgrund der zufälligen Ausgabe.

Importbefehle
finden Sie unter
[2nd](#) [catalog].

random()

Modul: random

[\[math\]](#) Modul

Syntax: random()

2:random...

Random

2:random()

Beschreibung: Ergibt eine Fließkommazahl zwischen 0 und 1,0. Diese Funktion verwendet keine Argumente.

Beispiel:

```
>>>from random import *
>>>random()
0.5381466990230621
```

[\[Fns...\]](#) > Modul

2:random...

Random

2:random()

Alternatives Beispiel:

```
>>>import random
>>>random.random()
0.2695098437037318
```

[\[2nd\]](#) [\[catalog\]](#)

Die Ergebnisse variieren aufgrund der zufälligen Ausgabe.

Importbefehle

finden Sie unter

[\[2nd\]](#) [\[catalog\]](#).

random.function

Modul: random

[\[2nd\]](#) [\[catalog\]](#)

Syntax: random.function

Beschreibung: Nach dem Import von random verwenden, um auf eine Funktion im Modul random zuzugreifen.

Beispiel:

```
>>>import random
>>>random.randint(1,15)
2
```

Die Ergebnisse variieren aufgrund der zufälligen Ausgabe.

randrange(start,stop,step)

Modul: random

[math](#) Modul
2:random...
Random
6:randrange
(
start,stop,step)

Syntax: randrange(start,stop,step)

Beschreibung: Ergibt eine Zufallszahl zwischen Start und Stopp mit definierter Schrittgröße.

Beispiel:

```
>>>from random import *  
>>>randrange(10,50,2)  
12
```

[math](#) Modul
2:random...
Random
6:randrange
(
start,stop,step)

Alternatives Beispiel:

```
>>>import random  
>>>random.randrange(10,50,2)  
48
```

Die Ergebnisse variieren aufgrund der zufälligen Ausgabe.

[2nd](#) [\[catalog\]](#)

Importbefehle
finden Sie
unter
[2nd](#)[\[catalog\]](#).

range(start,stop,step)

Modul: Integriert

[2nd](#) [\[catalog\]](#)

Syntax: range(start,stop,step)

Beschreibung: Verwenden Sie die Funktion „range“, um eine Zahlenfolge zurückzugeben. Alle Argumente sind optional. Die Standardwerte sind 0 für „start“ und 1 für „step“; bei „stop“ endet die Folge.

Beispiel:

```
>>> x = range(2,10,3)  
>>> for i in x  
... print(i)  
...  
...  
2  
5  
8
```

.real

Modul: Built-in

[2nd](#) [\[catalog\]](#)

Syntax: var.real

Beschreibung: Bestimmt den Realteil einer angegebenen Variablen eines komplexen Zahlentyps.

Beispiel:

```
>>>a=complex(4,5)
>>>a.real
4
>>>a.imag
5
```

var=recall_list("name") 1-6

Modul: ti_system

[2nd] [catalog]

Syntax: var=recall_list("name") 1-6

[2nd][rcI]

Beschreibung: Rufen Sie eine vordefinierte Betriebssystem-Liste auf. Die Länge der Liste muss kleiner als oder gleich 100 sein.

ti_system
4:var=recall_
list()

Argument: "name"

[Fns...]>Modul
oder [math]
4:ti_system
4:var=recall_
list()

Für L1-L6 im Betriebssystem

1-6

"1" - "6"

'1' - '6'

Für benutzerdefinierte Betriebssystem-Liste „name“

----- Max. 5 Zeichen, Zahlen oder Buchstaben, mit Buchstaben beginnend, Buchstaben müssen groß geschrieben werden.

Importbefehle
finden Sie
unter [2nd]
[catalog] oder
im Menü des
Moduls
ti_system.

Beispiele:

"ABCDE"

"R12"

"L1" ist ein benutzerdefiniertes L1 und nicht mit dem Betriebssystem-L1 identisch

Zur Erinnerung: Python ist doppelt genau. Python unterstützt mehr Ziffern als das Betriebssystem.

Beispiel:

Beispielprogramm:

Erstellen Sie eine Liste im Betriebssystem.
LIST={1,2,3}

Führen Sie die Python-App aus.
Erstellen Sie ein neues Programm.

```
import ti_system as *  
xlist=recall_list("LIST")  
print xlist
```

Führen Sie das Programm AA aus.
Die Shell zeigt die Ausgabe an.

```
[1.0, 2.0, 3.0]
```

var=recall_RegEQ()

Modul: ti_system

[2nd] [catalog]

Syntax: var=recall_RegEQ()

[2nd] [rc1]

Beschreibung: Rufen Sie die Variable RegEQ aus dem CE-Betriebssystem auf. Bevor RegEQ in der Python-App aufgerufen werden kann, muss die Regressionsgleichung im Betriebssystem berechnet werden.

ti_system
4:var=recall_RegEQ()

Beispiel:

[Fns...]>Modul
oder [math]
4:ti_system
4:var=recall_RegEQ()

Siehe Beispielprogramm: [REGEQ1](#).

Importbefehle
finden Sie unter
[2nd] [catalog]
oder im Menü
des Moduls
ti_system.

.remove(x)

Modul: Built-in

[2nd] [list]

Syntax: listname.remove(item)

List
7:remove(x)

Beschreibung: Die Methode remove() entfernt die erste Instanz eines Elements aus einer Folge.

[2nd] [catalog]

Beispiel:

```
>>>listA = [2,4,6,8,6]
>>>listA.remove(6)
>>>print(listA)
[2,4,8,6]
```

[Fns...] > List
7:remove(x)

return

Modul: Built-in

[\[2nd\]](#) [\[catalog\]](#)

Syntax: return expression

Beschreibung: Eine return-Anweisung definiert den von einer Funktion erzeugten Wert. Python-Funktionen geben standardmäßig „None“ zurück. Siehe auch: def function ():

```
[Fns...] > Func  
1:def function():
```

Beispiel:

```
>>> def f(a,b):  
...return a*b  
...  
...  
>>> f(2,3)  
6
```

```
[Fns...] > Func  
2:return
```

.reverse()

Modul: Built-in

[\[2nd\]](#) [\[catalog\]](#)

Syntax: listname.reverse()

Beschreibung: Gibt die Reihenfolge der Elemente in einer Folge zurück.

Beispiel:

```
>>>list1=[15,-32,4]  
>>>list1.reverse()  
>>>print(list1)  
[4,-32,15]
```

round()

Modul: Integriert

[\[2nd\]](#) [\[catalog\]](#)

Syntax: round(number, digits)

Beschreibung: Verwenden Sie die Rundungsfunktion, um eine Fließkommazahl auf die angegebenen Ziffern gerundet zurückzugeben. Die Standardziffer ist 0, wobei die nächste Ganzzahl angegeben wird.

Beispiel:

```
>>>round(23.12456)  
23  
>>>round(23.12456, 3)  
23.125
```

scatter(xlist,ylist,"mark")**Modul:** ti_plotlib[\[2nd\]](#)[catalog]**Syntax:** plt.scatter(xlist,ylist,"mark")[Fns...]>Modul
oder [\[math\]](#)
5:ti_plotlib...>
Draw
4:scatter()**Beschreibung:** Eine Folge geordneter Paare aus (xlist,ylist) wird unter Angabe des Markierungsstils geplottet. Linienstärke und -größe werden unter Verwendung von plt.pen() eingestellt.

xlist und ylist müssen echte Fließkommazahlen sein und Listen müssen die gleiche Dimension haben.

Importbefehle
finden Sie unter
[\[2nd\]](#)[catalog]
oder im Menü
ti_plotlib Setup.**Argument:**

„mark“ ist das Markierungszeichen, wie nachstehend erläutert:

-
- o **Gefüllter Punkt**
 (Standard)
 - + **Kreuz**
 - x **x**
 - . **Pixel**
-

Beispiel:Siehe Beispielprogramm: [LINREGR](#).

seed()

Modul: random

[math](#) Modul

Syntax: seed() oder seed(x), wobei x eine Ganzzahl ist

2:random...

Random

7:seed()

Beschreibung: Zufallszahlengenerierung initialisieren.

Beispiel:

[Fns...] >

Modul

2:random...

Random

7:seed()

```
>>>from random import *
>>>seed(12)
>>>random()
0.9079708720366826
>>>seed(10)
>>>random()
0.9063990882481896
>>>seed(12)
>>>random()
0.9079708720366826
```

[2nd](#) [catalog]

Die Ergebnisse variieren aufgrund der zufälligen Ausgabe.

Importbefehle

finden Sie

unter

[2nd](#) [catalog].

set(sequence)

Modul: Built-in

[2nd](#) [catalog]

Syntax: set(sequence)

Beschreibung: Gibt eine Folge als Satz zurück. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

Beispiel:

```
>>> print(set("84CE"))
{'E', '8', '4', 'C'}
```

show_plot() [display >](#) [\[clear\]](#)

Modul: ti_plotlib

[\[2nd\]](#) [\[catalog\]](#)

Syntax: plt.show_plot() [display >](#) [\[clear\]](#)

[Fns...]>Modul
oder [\[math\]](#)
5:ti_plotlib...>
Setup
9:show_plot

Beschreibung: Führt die Anzeige des Plots aus, wie im Programm eingerichtet.

show_plot() muss hinter allen Objekten für das Plot-Setup platziert werden. Die Reihenfolge der Plotting-Objekte im Programm wird durch die Reihenfolge im Setup-Menü vorgeschlagen.

[Fns...]>Modul
oder [\[math\]](#)
5:ti_plotlib... >
Draw
9:show_plot()

Um die Hilfe für die Plotting-Vorlage aus dem Dateimanager heraus anzuzeigen, wählen Sie [New] ([zoom]) und dann [Types] ([zoom]), um den Programmtyp „Plotting (x,y) & Text“ auszuwählen.

Nach Ausführung des Programms wird die Plotting-Anzeige durch Drücken von [\[clear\]](#) gelöscht, um zur Shell-Eingabeaufforderung zurückzukehren.

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#)
oder im Menü
ti_plotlib Setup.

Beispiel:

Siehe Beispielprogramme: [COLORLIN](#) oder [GRAPH](#).

sin()

Modul: math

[sin](#) 3:sin()

Syntax: sin()

Beschreibung: Ergibt den Sinus von x. Das Argument Winkel wird im Bogenmaß (Radian) ausgegeben.

[2nd](#) [\[catalog\]](#)

Beispiel:

```
>>>from math import *
>>>sin(pi/2)
1.0
```

[Fns...]> Modul
1:math... > Trig
3:sin()

Importbefehle
finden Sie unter
[2nd](#) [\[catalog\]](#).

sleep(seconds)

Modul: ti_system; time

[2nd](#) [\[catalog\]](#)

Syntax: sleep(seconds)

Beschreibung: Das Programm ruht, bis die in Sekunden angegebene Zeit verstrichen ist. Das Argument Sekunden ist eine Fließkommazahl.

[2nd](#) [\[rc1\]](#)
ti_system
A:sleep()

Beispiel:

Beispielprogramm:

```
from time import *
a=monotonic()
sleep(15)
b=monotonic()
print(b-a)
```

[Fns...]>Modul
oder [math](#)
4:ti_system
A:sleep()

[Fns...]>Modul
oder [math](#)
3:time
2:sleep()

Starten Sie das Programm TIME
>>>15.0

Importbefehle
finden Sie
unter [2nd](#)
[\[catalog\]](#) oder
im Menü des
Moduls
ti_system.

.sort()

Modul: Built-in

[2nd](#) [\[list\]](#)

Syntax: listname.sort()

(über [stat](#))
List A:.sort()

Beschreibung: Die Methode sortiert eine Liste. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

[2nd](#) [\[catalog\]](#)

Beispiel:

[Fns...] >
List
A:sort()

```
>>>listA=[4,3,6,2,7,4,8,9,3,5,4,6]
>>>listA.sort()
>>>print(listA) #listA wird in eine sortierte Liste
umgewandelt
[2,3,3,4,4,4,5,6,6,7,8,9]
```

sorted()

Modul: Built-in

[2nd](#) [\[list\]](#) (über
[stat](#)) List

Syntax: sorted(sequence)

O:sorted()

Beschreibung: Gibt eine sortierte Liste aus einer Folge zurück.

[2nd](#) [\[catalog\]](#)

Beispiel:

```
>>>listA=[4,3,6,2,7,4,8,9,3,5,4,6]
>>>sorted(listA)
[2,3,3,4,4,4,5,6,6,7,8,9]
>>>print(listA) #listA wurde nicht geändert
[4,3,6,2,7,4,8,9,3,5,4,6]
```

[Fns...] > List
O:sorted()

.split(x)

Modul: Built-in

[2nd](#) [\[catalog\]](#)

Syntax: var.split(x)

Beschreibung: Die Methode gibt eine Liste mit dem angegebenen Trennzeichen zurück. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

Beispiel:

```
>>> a="red,blue,green"
>>> a.split(",")
['red', 'blue', 'green']
```

sqrt()

Modul: math

[math](#) Modul
1:math 3:sqrt
()

Syntax: sqrt(x)

Beschreibung: Gibt die Quadratwurzel von x zurück.

Beispiel:

[2nd](#) [\[catalog\]](#)

```
>>>from math import *
>>>sqrt(25)
5.0
```

[Fns...] >
Modul 1:math
3:sqrt()

Importbefehle
finden Sie
unter
[2nd](#)
[\[catalog\]](#)[\[format\]](#).

store_list("name",var) 1-6

Modul: ti_system

[2nd] [catalog]

Syntax: store_list("name",var) 1-6

[2nd] [rc1]

Beschreibung: Speichert eine Liste von der Ausführung eines Python-Skripts in eine Betriebssystem-Listenvariable „name“, wobei var eine definierte Python-Liste ist. Die Länge der Liste muss kleiner als oder gleich 100 sein.

ti_system
3:var=store_list
()

Argument: "name"

[Fns...]>Modul
oder [math]
4:ti_system
3:var=store_list
()

Für L1-L6 im Betriebssystem

1-6

"1" - "6"

'1' - '6'

Importbefehle
finden Sie unter
[2nd] [catalog]
oder im Menü
des Moduls
ti_system.

Für benutzerdefinierte Betriebssystem-Liste „name“

----- Max. 5 Zeichen, Zahlen oder Buchstaben, mit Buchstaben beginnend, Buchstaben müssen groß geschrieben werden.

Beispiele:

"ABCDE"

"R12"

"L1" ist ein benutzerdefiniertes L1 und nicht mit dem Betriebssystem-L1 identisch

Zur Erinnerung: Python ist doppelt genau, d. h. es werden mehr Ziffern als im Betriebssystem unterstützt.

Beispiel:

```
>>>a=[1,2,3]
>>>store_list("1",a)
>>>
```

Beenden Sie die Python-App und drücken Sie [2nd][L1] (über [1]) und [enter] im Hauptbildschirm, um die Liste [L1] als {1 2 3} anzuzeigen.

str()

Modul: Built-in

[2nd](#) [\[catalog\]](#)

Syntax: str(argument)

Beschreibung: Konvertiert "argument" in einen String.

[Fns...]

Beispiel:

> Type

3 :str()

```
>>>x=2+3
>>>str(x)
'5'
```

sum()

Modul: Built-in

[2nd](#) [\[list\]](#) (über

[stat\]](#) List

Syntax: sum(sequence)

9:sum()

Beschreibung: Gibt die Summe der Elemente in einer Folge zurück.

[2nd](#) [\[catalog\]](#)

Beispiel:

```
>>>listA=[2,4,6,8,10]
>>>sum(listA)
30
```

[Fns...] > List

9:sum()

tan()**Modul:** math[\[sin\]](#) 5:tan()**Syntax:** tan(x)**Beschreibung:** Liefert den Tangens von x. Das Winkelargument wird in radians (Bogenmaß) ausgegeben.[Fns...] > Modul
1:math... > Trig
5:tan()**Beispiel:**

```
>>>from math import *
>>>tan(pi/4)
1.0
```

[\[2nd\]](#) [\[catalog\]](#)Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#).**text_at(row,"text","align")****Modul:** ti_plotlib[\[2nd\]](#) [\[catalog\]](#)**Syntax:** plt.text_at(row,"text","align")**Beschreibung:** "text" wird im Plotting-Bereich angezeigt, wie unter "align" angegeben.[Fns...]>Modul oder
[\[math\]](#)
5:ti_plotlib...>
Draw
0:text_at()

Zeile	Ganzzahl (1 bis 12)
"text"	Der String wird abgeschnitten, wenn er zu lang ist
"align"	"left" (Standard) "center" "right"
optional	1 löscht Zeile vor dem Text (Standard) 0 Zeile wird nicht gelöscht

Importbefehle
finden Sie unter
[\[2nd\]](#) [\[catalog\]](#) oder
im Menü
ti_plotlib Setup.**Beispiel:**Siehe Beispielprogramm: [DASH1](#).

time.function

Modul: Built-in

[\[2nd\]](#) [\[catalog\]](#)

Syntax: time.function

Beschreibung: Nach dem Import von time verwenden, um auf eine Funktion im Modul time zuzugreifen.

Beispiel:

Siehe: [\[Fns...\]>Modul: time and ti_system modules.](#)

title("title")

Modul: ti_plotlib

[\[2nd\]](#) [\[catalog\]](#)

Syntax: plt.title("title")

[Fns...]>Modul
oder [\[math\]](#)
5:ti_plotlib...>
Setup
8:title()

Beschreibung: "title" wird mittig in der obersten Zeile des Fensters angezeigt. Der Titel wird abgeschnitten, wenn er zu lang ist.

Beispiel:

Siehe Beispielprogramm: [COLORLIN.](#)

Importbefehle
finden Sie
unter [\[2nd\]](#)
[\[catalog\]](#) oder
im Menü
ti_plotlib
Setup.

ti_hub.function

Modul: ti_hub

[\[2nd\]](#) [\[catalog\]](#)

Syntax: ti_hub.function

Beschreibung: Nach dem Import von ti_hub verwenden, um auf eine Funktion im Modul ti_hub zuzugreifen.

Beispiel:

Siehe: [\[Fns...\]>Modul: ti_hub module](#).

ti_system.function

Modul: ti_system

[\[2nd\]](#) [\[catalog\]](#)

Syntax: ti_system.function

Beschreibung: Nach dem Import von ti_system verwenden, um auf eine Funktion im Modul ti_system zuzugreifen.

Beispiel:

```
>>> # Shell Reinitialized
>>> import ti_system
>>> ti_system.disp_at(6,8,"texte")
```

```
texte>>>|
```

#wird in Zeile 6, Spalte 8 mit der Shell-Eingabeaufforderung wie abgebildet angezeigt.

Wahr

Schlüsselwort

[2nd](#) [[test](#)]
(über [math](#))

Beschreibung: Gibt True zurück, wenn die ausgeführte Anweisung True ist. „True“ steht für einen wahren Wert bei Objekten des Typs „bool“.

[2nd](#) [[catalog](#)]

Beispiel:

```
>>>64>=32  
Wahr
```

[Fns...] > Ops
A:True

[a A #]

trunc()

Modul: math

[math](#) Modul

Syntax: trunc(x)

1:math...
0:trunc()

Beschreibung: Gibt den Zahlenwert x zurück, abgeschnitten auf eine Ganzzahl.

[2nd](#) [[catalog](#)]

Beispiel:

```
>>>from math import *  
>>>trunc(435.867)  
435
```

[Fns...] > Modul
1:math...
0:trunc()

Importbefehle
finden Sie unter
[2nd](#) [[catalog](#)].

try:

Schlüsselwort

[2nd](#) [[catalog](#)]

Beschreibung: Verwenden Sie „try code block“, um den Codeblock auf Fehler zu testen. Wird auch zusammen mit „except“ und „finally“ verwendet. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

tuple(sequence)

Modul: Built-in

[2nd](#) [\[catalog\]](#)

Syntax: tuple(sequence)

Beschreibung: Konvertiert eine Folge in ein Tupel um. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

Beispiel:

```
>>>a=[10,20,30]
>>>tuple(a)
(10,20,30)
```

type()

Modul: Built-in

[2nd](#) [\[catalog\]](#)

Syntax: type(object)

[Fns...]>Type>6:type
()

Beschreibung: Gibt die Typ des Objekts zurück.

Beispiel:

```
>>>a=1.25
>>>print(type(a))
<class 'float'>
>>>b=100
>>>print(type(b))
<class 'int'>
>>>a=10+2j
>>>print(type(c))
<class 'complex'>
```

U

uniform(min,max)

Modul: random

[math](#) Modul

Syntax: uniform(min,max)

2:random...

Random

3:uniform

(min,max)

Beschreibung: Liefert eine Zufallszahl x (float) zurück, wobei $\min \leq x \leq \max$.

Beispiel:

```
>>>from random import *
>>>uniform(0,1)
0.476118
>>>uniform(10,20)
16.2787
```

[2nd](#) [catalog](#)

Die Ergebnisse variieren aufgrund der zufälligen Ausgabe.

[Fns...] > Modul

2:random...

Random

3:uniform

(min,max)

Importbefehle
finden Sie unter

[2nd](#) [catalog](#).

W

wait_key()

Modul: ti_system

[\[2nd\]](#) [\[catalog\]](#)

Syntax: wait_key()

Beschreibung: Gibt einen kombinierten Tastencode zurück, der sich aus der gedrückten Taste und [\[2nd\]](#) und/oder [\[alpha\]](#) zusammensetzt. Die Methode wartet, bis eine Taste gedrückt wird, bevor sie zum Programm zurückkehrt.

Beispiel:

Siehe: [\[Fns...\]>Modul: time and ti_system modules.](#)

Siehe: [Keypad mapping for wait_key\(\)](#)

while condition:

Schlüsselwort

[\[Fns...\]](#) Ctl

Syntax: while condition:

8:while condition:

Beschreibung: Führt die Anweisungen im folgenden Codeblock aus, bis „condition“ als „False“ ausgewertet wird.

[\[2nd\]](#) [\[catalog\]](#)

Beispiel:

```
>>> x=5
>>> while x<8:
... x=x+1
... print(x)
...
...
6
7
8
```

window(xmin,xmax,ymin,ymax)

Modul: ti_plotlib

[2nd] [catalog]

Syntax: plt.window(xmin,xmax,ymin,ymax)

[Fns...]>Modul
oder [math]
5:ti_plotlib...>
Setup
4:window()

Beschreibung: Definiert das Plotting-Fenster, indem es das angegebene horizontale Intervall (xmin, xmax) und das angegebene vertikale Intervall (ymin, ymax) auf den zugeordneten Plotting-Bereich (Pixel) abbildet.

Diese Methode muss ausgeführt werden, bevor andere Befehle des Moduls ti_plotlib ausgeführt werden.

Importbefehle
finden Sie
unter [2nd]
[catalog] oder
im Menü
ti_plotlib
Setup.

Die ti_plotlib-Eigenschaften vars, xmin, xmax, ymin, ymax werden auf die Argumentwerte aktualisiert. Die Standardwerte sind (-10, 10, -6.56, 6.56).

Beispiel:

Siehe Beispielprogramm: [GRAPH](#).

with

Schlüsselwort

[2nd] [catalog]

Beschreibung: Weitere Einzelheiten finden Sie in der Python-Dokumentation.

X

xmax **default** **10.00**

Modul: ti_plotlib

[2nd] [catalog]

Syntax: plt.xmax default 10.00

[Fns...]>Modul oder
[math]

Beschreibung: Festgelegte Variable für
Fensterargumente, definiert als plt.xmax.

5:ti_plotlib...>
Properties
2:xmax

Standardwerte:

xmin default -10.00

xmax default 10.00

ymin default -6.56

ymax default 6.56

Importbefehle
finden Sie unter
[2nd] [catalog] oder
im Menü
ti_plotlib Setup.

Beispiel:

Siehe Beispielprogramm: [GRAPH](#).

xmin **default** **-10.00**

Modul: ti_plotlib

[2nd] [catalog]

Syntax: plt.xmin **default** **-10.00**

[Fns...]>Modul oder
[math]

Beschreibung: Festgelegte Variable für
Fensterargumente, definiert als plt.xmin.

5:ti_plotlib...>
Properties
1:xmin

Standardwerte:

xmin **default -10.00**

xmax **default 10.00**

ymin **default -6.56**

ymax **default 6.56**

Importbefehle
finden Sie unter
[2nd] [catalog] oder
im Menü
ti_plotlib Setup.

Beispiel:

Siehe Beispielprogramm: [GRAPH](#).

yield**Schlüsselwort**[\[2nd\]](#) [\[catalog\]](#)

Beschreibung: Verwenden Sie „yield“ (ergibt), um eine Funktion zu beenden. Es wird ein Generator ausgegeben. Weitere Einzelheiten finden Sie in der Python-Dokumentation.

ymax **default** **6.56****Modul:** `ti_plotlib`[\[2nd\]](#) [\[catalog\]](#)**Syntax:** `plt.ymax` **default** **6.56**`[Fns...]>Modul` oder [\[math\]](#)

Beschreibung: Festgelegte Variable für Fensterargumente, definiert als `plt.ymax`.

`5:ti_plotlib...>`

Properties

`4:ymax`**Standardwerte:**

`xmin` **default -10.00**`xmax` **default 10.00**`ymin` **default -6.56**`ymax` **default 6.56**

Importbefehle finden Sie unter [\[2nd\]](#) [\[catalog\]](#) oder im Menü `ti_plotlib` Setup.

Beispiel:Siehe Beispielprogramm: [GRAPH](#).

ymin **default** **-6.56**

Modul: ti_plotlib

[2nd] [catalog]

Syntax: plt.ymin **default** **-6.56**

[Fns...]>Modul oder
[math]

Beschreibung: Festgelegte Variable für
Fensterargumente, definiert als plt.ymin.

5:ti_plotlib...>
Properties
3:ymin

Standardwerte:

xmin **default -10.00**

xmax **default 10.00**

ymin **default -6.56**

ymax **default 6.56**

Importbefehle
finden Sie unter
[2nd] [catalog] oder
im Menü
ti_plotlib Setup.

Beispiel:

Siehe Beispielprogramm: [GRAPH](#).

Sonderzeichen

@

Operator

[alpha](#) [[θ](#)]
(über [3](#))

Beschreibung: Decorator – Einzelheiten entnehmen Sie bitte der allgemeinen Python-Dokumentation.

[2nd](#) [[catalog](#)]

<<

Operator

[2nd](#) [[catalog](#)]

Syntax: x<<n

Beschreibung: Bitweises Verschieben nach links um n Bits.

>>

Operator

[2nd](#) [[catalog](#)]

Syntax: x>>n

Beschreibung: Bitweises Verschieben nach rechts um n Bits.

|

Operator

[2nd](#) [[catalog](#)]

Syntax: x|y

Beschreibung: Bitweises oder.

&

Operator

[2nd](#) [[catalog](#)]

Syntax: x&y

Beschreibung: Bitweises und.

^

Operator

[2nd](#) [\[catalog\]](#)

Syntax: x^y

Beschreibung: Bitweises exklusives oder.

~

Operator

[2nd](#) [\[catalog\]](#)

Syntax: $\sim x$

Beschreibung: Bitweises nicht; die Bits von x invertiert.

$x \leq y$

Operator

`[math]`

Syntax: $x \leq y$

1:math > Ops

7:x<=y

Beschreibung: Vergleich; x ist kleiner oder gleich y.

Beispiel:

`[2nd]` `[catalog]`

```
>>>2<=5
```

```
Wahr
```

```
>>>3<=0
```

```
Falsch
```

`[Fns...]` > Ops

7:x<=y

[a A #]

$x < y$

Operator

`[math]`

Syntax: $x < y$

1:math > Ops

6:x<y

Beschreibung: Vergleich; x ist strikt kleiner als y.

Beispiel:

`[2nd]` `[catalog]`

```
>>>6<10
```

```
Wahr
```

```
>>>12<-15
```

```
Falsch
```

`[Fns...]` > Ops

6:x<y

[a A #]

$x \geq y$

Operator

`math`

Syntax: $x \geq y$

1: `math >` Ops

5: `x >= y`

Beschreibung: Vergleich; x ist größer oder gleich y.

Beispiel:

`2nd` [catalog]

```
>>>35>=25
```

```
Wahr
```

```
>>>14>=65
```

```
Falsch
```

[Fns...] > Ops

5: `x >= y`

[a A #]

$x > y$

Operator

`math`

Syntax: $x > y$

1: `math >` Ops

4: `x > y`

Beschreibung: Vergleich; x ist strikt größer als y.

Beispiel:

`2nd` [catalog]

```
>>>35>25
```

```
Wahr
```

```
>>>14>65
```

```
Falsch
```

[Fns...] > Ops

4: `x > y`

[a A #]

x!=y

Operator

[math](#)

Syntax: x!=y

1:math > Ops
3:x!=y

Beschreibung: Vergleich; x ist ungleich zu y.

Beispiel:

[2nd](#) [[catalog](#)]

```
>>>35!=25  
Wahr  
>>>14!=10+4  
Falsch
```

[Fns...] > Ops
3:x!=y

[a A #]

x==y

Operator

[math](#)

Syntax: x==y

1:math > Ops
2:x==y

Beschreibung: Vergleich; x ist gleich y.

Beispiel:

[2nd](#) [[catalog](#)]

```
>>>75==25+50  
Wahr  
>>>1/3==0.333333  
Falsch  
>>>1/3==0.3333333 #gleich dem gespeicherten Python-Wert  
Wahr
```

[Fns...] > Ops
2:x==y

[a A #]

x=y

Operator

[sto →](#)

Syntax: x=y

Beschreibung: y ist gespeichert in der Variablen x

[math](#)

1:math > Ops

1:x=y

Beispiel:

```
>>>A=5.0
>>>print(A)
5.0
>>>B=2**3 #Verwenden Sie [ ^ ] auf der Tastatur für **
>>>print(B)
8
```

[2nd](#) [\[catalog\]](#)

[\[Fns...\]](#) > Ops

1:x=y

[\[a A #\]](#)

Trennzeichen

[2nd](#) [\[catalog\]](#)

Beschreibung: Umgekehrter Schrägstrich (Backslash).

[\[a A #\]](#)

\t

Trennzeichen

[2nd](#) [\[catalog\]](#)

Beschreibung: Tabulatorschritt zwischen Strings oder Zeichen.

\n

Trennzeichen

[2nd](#) [\[catalog\]](#)

Beschreibung: Neue Zeile zur übersichtlichen Darstellung des Strings auf dem Bildschirm.

' '

Trennzeichen

[2nd](#) [\[mem\]](#)
(über [+](#))

Beschreibung: Es werden zwei einzelne
Anführungszeichen eingefügt.

Beispiel:

[2nd](#) [\[catalog\]](#)

```
>>>eval('a+10')  
17
```

[a A #]

" "

Trennzeichen

[alpha](#) [\["\]](#)
(über [+](#))

Beschreibung: Es werden zwei doppelte
Anführungszeichen eingefügt.

Beispiel:

[2nd](#) [\[catalog\]](#)

```
>>>print("Ok")
```

[a A #]

Anhang

[Ausgewählte Inhalte der in TI-Python integrierten Funktionen, Schlüsselwörter und Module](#)

[Tastaturbelegung für wait_key\(\)](#)

Ausgewählte Inhalte der in TI-Python integrierten Funktionen, Schlüsselwörter und Module

Integriert

Integriert	Integriert	Integriert
<code>__name__</code>	<code>abs</code> -- <function>	<code>BaseException</code> -- <class 'BaseException'>
<code>__build_class__</code> -- <function>	<code>all</code> -- <function>	<code>ArithmeticError</code> -- <class 'ArithmeticError'>
<code>__import__</code> -- <function>	<code>any</code> -- <function>	<code>AssertionError</code> -- <class 'AssertionError'>
<code>__repl_print__</code> -- <function>	<code>bin</code> -- <function>	<code>AttributeError</code> -- <class 'AttributeError'>
<code>bool</code> -- <class 'bool'>	<code>callable</code> -- <function>	<code>EOFError</code> -- <class 'EOFError'>
<code>bytes</code> -- <class 'bytes'>	<code>chr</code> -- <function>	<code>Exception</code> -- <class 'Exception'>
<code>bytearray</code> -- <class 'bytearray'>	<code>dir</code> -- <function>	<code>GeneratorExit</code> -- <class 'GeneratorExit'>
<code>dict</code> -- <class 'dict'>	<code>divmod</code> -- <function>	<code>ImportError</code> -- <class 'ImportError'>
<code>enumerate</code> -- <class 'enumerate'>	<code>eval</code> -- <function>	<code>IndentationError</code> -- <class 'IndentationError'>
<code>filter</code> -- <class 'filter'>	<code>exec</code> -- <function>	<code>IndexError</code> -- <class 'IndexError'>
<code>float</code> -- <class 'float'>	<code>getattr</code> -- <function>	<code>KeyboardInterrupt</code> -- <class 'KeyboardInterrupt'>
<code>int</code> -- <class 'int'>	<code>setattr</code> -- <function>	<code>ReloadException</code> -- <class

Integriert	Integriert	Integriert
		'ReloadException'>
list -- <class 'list'>	globals -- <function>	KeyError -- <class 'KeyError'>
map -- <class 'map'>	hasattr -- <function>	LookupError -- <class 'LookupError'>
memoryview -- <class 'memoryview'>	hash -- <function>	MemoryError -- <class 'MemoryError'>
object -- <class 'object'>	help -- <function>	NameError -- <class 'NameError'>
property -- <class 'property'>	hex -- <function>	NotImplementedError -- <class 'NotImplementedError'>
range -- <class 'range'>	id -- <function>	OSError -- <class 'OSError'>
set -- <class 'set'>	input -- <function>	OverflowError -- <class 'OverflowError'>
slice -- <class 'slice'>	isinstance -- <function>	RuntimeError -- <class 'RuntimeError'>
str -- <class 'str'>	issubclass -- <function>	StopIteration -- <class 'StopIteration'>
super -- <class 'super'>	iter -- <function>	SyntaxError -- <class 'SyntaxError'>
tuple -- <class 'tuple'>	len -- <function>	SystemExit -- <class 'SystemExit'>
type -- <class 'type'>	locals -- <function>	TypeError -- <class 'TypeError'>
zip -- <class 'zip'>	max -- <function>	UnicodeError -- <class 'UnicodeError'>
classmethod -- <class 'classmethod'>	min -- <function>	ValueError -- <class 'ValueError'>
staticmethod -- <class 'staticmethod'>	next -- <function>	ZeroDivisionError -- <class 'ZeroDivisionError'>

Integriert	Integriert	Integriert
Ellipsis -- Ellipsis	oct -- <function>	
	ord -- <function>	
	pow -- <function>	
	print -- <function>	
	repr -- <function>	
	round -- <function>	
	sorted -- <function>	
	sum -- <function>	

Schlüsselwörter

Schlüsselwörter	Schlüsselwörter	Schlüsselwörter
False	elif	lambda
None	else	nonlocal
True	except	not
and	finally	or
as	for	pass
assert	from	raise
break	global	return
class	if	try
continue	import	while
def	in	with
del	is	yield

math

```
PYTHON SHELL
>>> import math
>>> dir(math)
['_name__', 'e', 'pi', 'sqrt',
'pow', 'exp', 'log', 'cos', 'sin',
'tan', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'copysign',
'fabs', 'floor', 'fmod', 'frexp',
'ldexp', 'modf', 'isfinite', 'i
sinf', 'isnan', 'trunc', 'radian
s', 'degrees']
>>> |
```

math	math	math
<code>__name__</code>	<code>acos</code> -- <function>	<code>frexp</code> -- <function>
<code>e</code> -- 2.71828	<code>asin</code> -- <function>	<code>ldexp</code> -- <function>
<code>pi</code> -- 3.14159	<code>atan</code> -- <function>	<code>modf</code> -- <function>
<code>sqrt</code> -- <function>	<code>atan2</code> -- <function>	<code>isfinite</code> -- <function>
<code>pow</code> -- <function>	<code>ceil</code> -- <function>	<code>isinf</code> -- <function>
<code>exp</code> -- <function>	<code>copysign</code> -- <function>	<code>isnan</code> -- <function>
<code>log</code> -- <function>	<code>fabs</code> -- <function>	<code>trunc</code> -- <function>
<code>cos</code> -- <function>	<code>floor</code> -- <function>	<code>radians</code> -- <function>
<code>sin</code> -- <function>	<code>fmod</code> -- <function>	<code>degrees</code> -- <function>
<code>tan</code> -- <function>		

random

```
PYTHON SHELL
>>> import random
>>> dir(random)
['__name__', 'seed', 'getrandbit
s', 'randrange', 'randint', 'cho
ice', 'random', 'uniform']
>>> |
```

Fns... a A # Tools Editor Files

random	random	random
<code>__name__</code>	<code>randint -- <function></code>	
<code>seed -- <function></code>	<code>choice -- <function></code>	
<code>getrandbits -- <function></code>	<code>random -- <function></code>	
<code>randrange -- <function></code>	<code>uniform -- <function></code>	

time

```
PYTHON SHELL
>>> import time
>>> dir(time)
['_name__', 'monotonic', 'sleep',
 'struct_time']
>>> |
```

Fns... a A # Tools Editor Files

time	time	time
<code>__name__</code>		
<code>monotonic</code>		
<code>sleep</code>		
<code>struc_time</code>		

ti_system

```
PYTHON SHELL
>>> import ti_system
>>> dir(ti_system)
['__name__', 'escape', 'recall_l
ist', 'store_list', 'recall_RegE
Q', 'wait_key', 'sleep', 'wait',
'disp_at', 'disp_clr', 'disp_wa
it', 'disp_cursor']
>>> |
```

Fns... a A # Tools|Editor|Files

ti_system	ti_system	ti_system
<code>__name__</code>	<code>recall_RegEQ</code>	<code>disp_at</code>
<code>escape</code>	<code>wait_key</code>	<code>disp_clr</code>
<code>recall_list</code>	<code>sleep</code>	<code>disp_wait</code>
<code>store_list</code>	<code>wait</code>	<code>disp_cursor</code>

ti_plotlib

```
PYTHON SHELL
>>> import ti_plotlib
>>> dir(ti_plotlib)
['lin_reg', '_strtest', 'escape',
 '_except', 'text_at', '_clipseg',
 'show_plot', 'tilocal', 'pen',
 '_sys', 'xmin', 'ymax', 'yscl',
 '_xy', '_rdelta', '_ydelta', 's
catter', 'a', '_pencolor', '_wri
te', 'b', '_xytest', 'window', '_
_mark', 'line', 'monotonic', '_n
umtest', 'ymin', 'tiplotlibExcep
tion', 'labels', 'cls', 'sqrt',
'xscl', 'axes', 'grid', '_sema',
'_pensize', 'plot', 'isnan', 'c
olor', 'title', '_xdelta', '_pen
style', '__name__', 'copysign',
'gr', 'xmax', 'sleep', 'auto_win
dow']
>>> |
Fns... a A # |Outils|Éditer|Script
```

ti_plotlib	ti_plotlib	ti_plotlib
<code>__name__</code>	a	grid
<code>lin_reg</code>	<code>_pencolor</code>	-pensize
<code>_strtest</code>	<code>_write</code>	<code>_sema</code>
<code>escape</code>	b	-pensize
<code>_except</code>	<code>_xytest</code>	plot
<code>text_alt</code>	window	isnan
<code>_clipseg</code>	<code>_mark</code>	color

ti_plotlib	ti_plotlib	ti_plotlib
show-plot	line	title
tilocal	monotonic	_xdelta
pen	_ntest	_penstyle
sys	ymin	copysign
xmin	tiplotlibException	gr
ymax	lables	xmax
yscl	cls	sleep
_xy	sqrt	auto_window
_rdelta	xscl	
_ydelta	axes	
scatter		

ti_hub

```
PYTHON SHELL
>>> import ti_hub
>>> dir(ti_hub)
['__name__', 'connect', 'disconnect', 'set', 'read', 'calibrate', 'range', 'version', 'about', 'isti', 'what', 'who', 'begin', 'wait', 'sleep', 'start', 'last_error', 'get', 'send', 'tihubException']
>>> |
```

Fns... a A # Tools Editor Files

ti_hub	ti_hub	ti_hub
__name__	version	last_error
connect	begin	sleep
disconnect	start	tihubException
set	about	wait
read	isti	get
calibrate	what	send
range	who	

ti_rover

```
PYTHON SHELL
>>> import ti_rover
>>> dir(ti_rover)
['motor_right', 'to_angle', 'to_xy', 'red_measurement', 'gray_measurement', 'rmovement', '_except', 'ranger_time', 'waypoint_prev', 'ti_hub', 'pathlist_time', 'to_polar', 'waypoint_eta', 'color_off', 'grid_m_unit', 'path_clear', 'green_measurement', 'waypoint_time', 'motors', 'backward', 'color_blink', 'motor_left', 'waypoint_heading', 'motor', 'gyro_measurement', 'wait_until_done', 'encoders_gyro_measurement', 'pathlist_distance', 'position', 'blue_measurement', 'forward', 'waypoint_distance', 'grid_origin', 'resume', 'path_done', 'disconnect_rv', 'backward_time', 'zero_gyro', '_rv_connected', 'stop', '_rv', 'stay', 'waypoint_xythdrn', 'ranger_measurement', 'left', 'pathlist_cmdnum', 'waypoint_y', 'waypoint_x', 'pathlist_y', 'pathlist_x', '_name_', 'right', 'color_rgb', 'pathlist_revs', 'color_measurement', 'pathlist_heading', 'forward_time', 'waypoint_revs']
>>> |
Fns... a R # Tools Editor Files
```

ti_rover	ti_rover	ti_rover
__name__	color_blink	_rv
motor_right	motor_left	stay

ti_rover	ti_rover	ti_rover
to_angle	waypoint_heading	waypoint_xythdrn
to_xy	_motor	ranger_measurement
red_mesurment	gyro_measutrmnt	left
rvmovement	wait_until_done	pathlist_cmdnum
gray_mesurment	encoders_gyro_measurement	waypoint_y
_excpt	pathlist_distance	waypoint-x
ti_hub	position	pathlist_y
waypoint_prev	blue_measurement	pathlist_x
pathlist_time	forward	right
waypoint_revs	waypoint_distance	color_rgb
to_polar	grid_origin	pathlist-revs
waypoint_eta	resume	color_measurement
color_off	path_done	tiroverException
grid_m_unit	disconnect_rv	forward_time
path_clear	backward_time	pathlist_heading
green_measurement	zero-gyro	
waypoint_time	_rv_connected	

ti_rover	ti_rover	ti_rover
motors	stop	
backward		

Tastaturbelegung für wait_key()

statplot #1 y=	73 48 73	tblset #2 window	72 75 72	format #3 zoom	46 87 46	calc #4 trace	90 59 90	table #5 graph	68 74 68
2nd	----	quit mode	69 64 69	ms del	10 11 10	< ←	2 14 2	> ▶	1 15 1
A-lock alpha	----	link X,T,θ,n	180 65 180	list stat	49 58 49	 ▲	3 ----- 3	 ▼	4 ----- 4
test A math	50 51 154	angle B apps	44 57 155	draw C prgm	45 47 156	distr vars	53 56 53	clear	9 9 9
matrix D x ⁻¹	182 55 157	sin ⁻¹ E sin	183 184 158	cos ⁻¹ F cos	185 186 159	tan ⁻¹ G tan	187 188 160	π H ^	132 181 161
√ I x ²	189 190 162	EE J ,	139 152 163	(K (133 236 164) L)	134 237 165	÷ M ÷	131 239 166
10 ^x N log	193 194 167	u O 7	149 249 168	v P 8	150 250 169	w Q 9	151 251 170	i R ×	130 135 171
e ^x S ln	191 192 172	L4 T 4	146 246 173	L5 U 5	147 247 174	L6 V 6	148 248 175	J W -	129 136 176
rcI X sto→	138 12 177	L1 Y 1	143 243 178	L2 Z 2	144 244 179	L3 θ 3	145 245 204	mem # +	128 54 203
off on	break ----- -----	catalog 0	142 62 153	i : .	141 238 198	ans ? (-)	140 197 202	entry solve enter	5 13 5

Allgemeine Informationen

Online-Hilfe

education.ti.com/eguide

Wählen Sie Ihr Land aus, um weitere Produktinformationen zu erhalten.

Kontakt mit TI Support aufnehmen

education.ti.com/ti-cares

Wählen Sie Ihr Land aus, um auf technische und sonstige Support-Ressourcen zuzugreifen.

Service- und Garantieinformationen

education.ti.com/warranty

Wählen Sie für Informationen zur Dauer und den Bedingungen der Garantie bzw. zum Produktservice Ihr Land aus.

Eingeschränkte Garantie. Diese Garantie hat keine Auswirkungen auf Ihre gesetzlichen Rechte.