

# TI-Nspire™ Python Programming Guidebook

## ***Important Information***

Except as otherwise expressly stated in the License that accompanies a program, Texas Instruments makes no warranty, either express or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding any programs or book materials and makes such materials available solely on an "as-is" basis. In no event shall Texas Instruments be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of the purchase or use of these materials, and the sole and exclusive liability of Texas Instruments, regardless of the form of action, shall not exceed the amount set forth in the license for the program. Moreover, Texas Instruments shall not be liable for any claim of any kind whatsoever against the use of these materials by any other party.

© 2021 Texas Instruments Incorporated

"Python" and the Python logos are trademarks or registered trademarks of the Python Software Foundation, used by Texas Instruments Incorporated with permission from the Foundation. Actual products may vary slightly from provided images.

# もくじ

<b>Pythonプログラミング</b> .....	<b>4</b>
Pythonモジュール .....	4
Pythonプログラムのモジュール化 .....	5
<b>Pythonワークスペース</b> .....	<b>6</b>
Pythonエディタ .....	6
Pythonシェル .....	10
<b>Pythonメニュー</b> .....	<b>13</b>
Actions(アクション)メニュー .....	14
Run(実行)メニュー .....	14
Tools(ツール)メニュー .....	14
Edit(編集)メニュー .....	15
Built-ins(組み込み)メニュー .....	16
Math(数学)メニュー .....	18
Random(乱数)メニュー .....	20
TI PlotLib(TI プロットライブラリ)メニュー .....	20
TI Hub(TI ハブ)メニュー .....	22
TI Rover(TI ローバー)メニュー .....	30
Complex Math(複素数)メニュー .....	36
Time(時間)メニュー .....	36
TI System(TI システム)メニュー .....	37
TI Draw(TI 描画)メニュー .....	38
TI Image(TI 画像)メニュー .....	40
Variables(変数)メニュー .....	42
<b>付録</b> .....	<b>43</b>
Pythonキーワード .....	44
Pythonキー対応 .....	45
例：Pythonプログラム .....	47
<b>一般情報</b> .....	<b>54</b>

# Pythonプログラミング

TI-Nspire™製品では、Python(パイソン)を使って次のことができます。

- PythonプログラムをTNSファイルに追加します。
- テンプレートを使ってPythonプログラムを作成します。
- 他のTI-Nspire™アプリとデータを共有します。
- TI-Innovator™ HubやTI-Innovator™ Roverを動かせます。

TI-Nspire™ Pythonは、マイクロコントローラーで実行するよう設計されたPython 3 標準ライブラリのサブセットであるMicroPythonに基づいています。元のMicroPythonは、TIで使用できるように調整されています。

**Note:** 基礎となる数学機能の違いにより、数値の結果は電卓の結果と異なる場合があります。

Pythonは、次のTI-Nspire™製品で使用できます。

グラフ電卓	デスクトップソフトウェア
TI-Nspire™ CX II	TI-Nspire™ CX Premium Teacher Software
TI-Nspire™ CX II CAS	TI-Nspire™ CX CAS Premium Teacher Software
TI-Nspire™ CX II-T	TI-Nspire™ CX Student Software
TI-Nspire™ CX II-T CAS	TI-Nspire™ CX CAS Student Software
TI-Nspire™ CX II-C	
TI-Nspire™ CX II-C CAS	

**Note:** ほとんど機能はグラフ電卓ビューとソフトウェアビューで同じですが、いくつかの違いが見られます。このガイドブックは、ソフトウェア上のグラフ電卓画面またはグラフ電卓そのものの画面を使っています。

## Pythonモジュール

TI-Nspire™ Pythonは、次のモジュール(特定の機能を持った部品)を持っています。

標準モジュール	TI モジュール
Math (数学)	TI PlotLib (ti_plotlib)
Random (乱数)	TI Hub (ti_hub)
Complex Math (複素数)	TI Rover (ti_rover)
Time (時間)	TI System (ti_system)
	TI Draw (ti_draw)
	TI Image (ti_image)

**Note:** 他のPython開発環境で作成されたPythonプログラムがある場合、TI-Nspire™ Pythonで実行するには、編集が必要な場合があります。モジュールは、TIモジュールと比較して、プログラム内で異なるメソッド、引数、メソッド順序を使う場合があります。一般に、任意のバージョンのPythonとPythonモジュールを使う場合は、互換性に注意してください。

Pythonプログラムを非TIプラットフォームからTIプラットフォームに、またはあるTI製品から別のTI製品に転送する場合、次の点に注意してください。

- コア言語機能と標準ライブラリ(Math, Randomなど)を使うプログラムは、変更せずに移植できます。
- PCまたはTIモジュール用のmatplotlibなどの、プラットフォーム固有のライブラリを使うプログラムは、別のプラットフォームでの実行前に編集する必要があります。これは、TIプラットフォーム間でも当てはまる場合があります。

Pythonの他のバージョンと同様、特定のモジュールに含まれる関数、メソッド、定数を使うには、インポートを含める必要があります。たとえば、Mathモジュールからcos()関数を実行するには、次のコマンドを使います。

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

メニューとその項目、説明については、Pythonメニュー(p.13)をご参照ください。

## Pythonプログラムのモジュール化

Pythonプログラムをモジュールとして保存するには、次のようにします。

- エディタでは、**Actions > Install as Python Module** (アクション>Pythonモジュールとしてインストール)を選択します。
- シェルでは、**Tools > Install as Python Module** (ツール>Pythonモジュールとしてインストール)を選択します。

選択すると、次のようになります。

- Python構文がチェックされます。
- ファイルが保存され、PyLibフォルダーに移動します。
- ファイルがモジュールとしてインストールされたことを確認するダイアログボックスが表示されます。
- ファイルが閉じられ、モジュールが使えるようになります。
- モジュール名は**More Modules**(その他のモジュール)メニューに追加されます。メニュー項目に**from <module> import \*((<モジュール>からインポート\*)**が表示されます。

モジュールを他の人と共有するときは、次のガイドラインに従うことをお勧めします。

- TNSファイルに1つのモジュールのみを保存します。
- モジュール名はTNSファイルの名前と一致します(たとえば、"my\_program"モジュールは"my\_program.tns"ファイルにあります)。
- Pythonエディタの前に、モジュールの目的、バージョン、関数を説明するメモページを追加します。
- ver()関数を使ってモジュールのバージョン番号を表示します。
- (オプション)ヘルプ関数を追加して、関数内のメソッドのリストを表示します。

# Pythonワークスペース

Pythonプログラミングには、PythonエディタとPythonシェル(Shell)の2つのワークスペース(作業場所)があります。

Pythonエディタ	Pythonシェル
<ul style="list-style-type: none"><li>• Pythonプログラムを作成、編集、保存</li><li>• 構文の強調表示と自動インデント(字下げ)</li><li>• 関数は引数で入力をガイドするインラインプロンプト(行内の入力要請)</li><li>• 有効な値を表示するツールチップ</li><li>• <code>vars</code>は現在のプログラムで定義されているグローバルユーザー変数と関数を一覧表示</li><li>• キーバッドのショートカット</li></ul>	<ul style="list-style-type: none"><li>• Pythonプログラムの実行</li><li>• 小さなコードフラグメントのテストに便利</li><li>• 以前の入力と出力を再利用するためシェル履歴と相互に作業</li><li>• <code>vars</code>はプログラムで定義されたグローバルユーザー変数を一覧表示</li></ul>

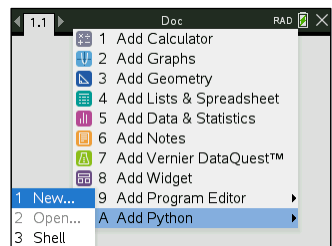
**Note:** 1つのプロブレムには、複数のPythonプログラムとシェルを作成することができます。

## Pythonエディタ

PythonエディタはPythonプログラムを作成、編集、保存できる場所です。

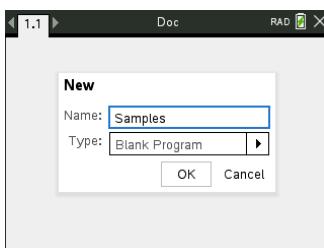
### Pythonエディタページの追加

現在のプロブレムに新規Pythonエディタページを追加するには、`menu`を押して、**Add Python > New** (Pythonを追加>新規)を選択します。

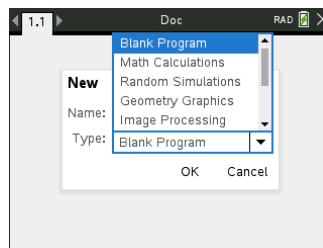


開いたダイアログボックスで、**Name:**はファイル名を入力します。**Type:**は空白のプログラムやテンプレートを選択します。

#### 空白のプログラム(Blank Program)



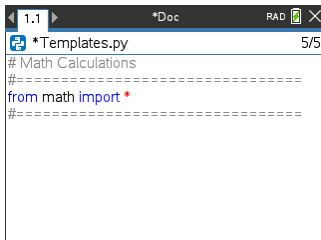
#### テンプレート



プログラムを作成するとPythonエディタが表示されます。テンプレートを選択した場合、必要なインポートステートメントが自動的に追加されます(以下を参照)。

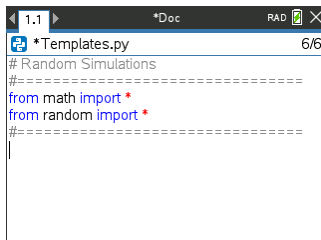
**Note:** 他のアプリと同じように、1つのTNSファイルに複数のプログラムを含めることができます。Pythonプログラムをモジュールとして使う場合、TNSファイルをPyLibフォルダーに保存できます。そのモジュールは、他のプログラムやドキュメントで使用できます。

### Math Calculations(数学計算)



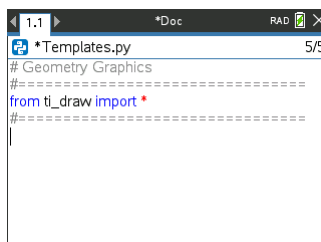
```
1.1 | *Doc | RAD | *Templates.py | 5/5
# Math Calculations
#=====
from math import *
#=====
```

### Random Simulations(乱数シミュレーション)



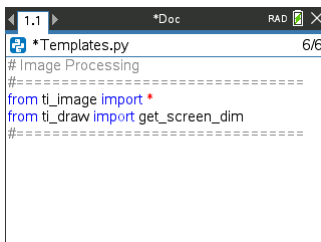
```
1.1 | *Doc | RAD | *Templates.py | 6/6
# Random Simulations
#=====
from math import *
from random import *
#=====
```

### Geometry Graphics(幾何グラフ)



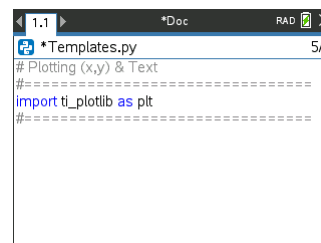
```
1.1 | *Doc | RAD | *Templates.py | 5/5
# Geometry Graphics
#=====
from ti_draw import *
#=====
```

### Image Processing(画像処理)



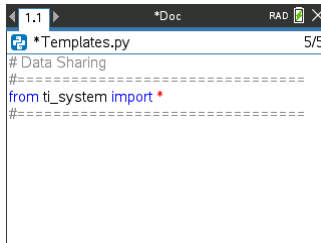
```
1.1 | *Doc | RAD | *Templates.py | 6/6
# Image Processing
#=====
from ti_image import *
from ti_draw import get_screen_dim
#=====
```

### Plotting(x, y)&Text(プロットとテキスト)



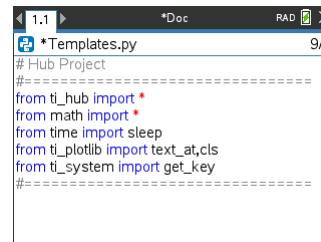
```
1.1 | *Doc | RAD | *Templates.py | 5/5
# Plotting (x,y) & Text
#=====
import ti_plotlib as plt
#=====
```

### Data Sharing(データ共有)



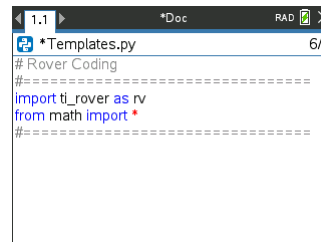
```
1.1 | *Doc | RAD | *Templates.py | 5/5
# Data Sharing
#=====
from ti_system import *
#=====
```

### Hub Project(TI-Innovator Hub)



```
1.1 | *Doc | RAD | *Templates.py | 9/9
# Hub Project
#=====
from ti_hub import *
from math import *
from time import sleep
from ti_plotlib import text_at_cls
from ti_system import get_key
#=====
```

### Rover Coding(TI-Roverコーディング)



```
1.1 | *Doc | RAD | *Templates.py | 6/6
# Rover Coding
#=====
import ti_rover as rv
from math import *
#=====
```

## Pythonプログラムを開く

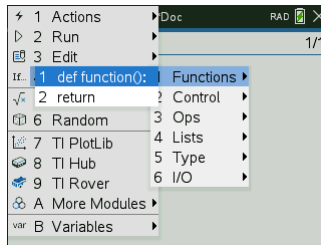
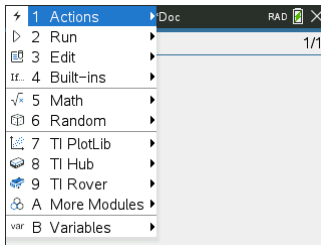
既存のPythonプログラムを開くには、**[doc]**を押してInsert > Add Python > Open (挿入>Pythonの追加>開く)を選択します。これにより、TNSファイルに保存されているプログラムのリストが表示されます。

プログラムの作成に使用されたエディタページが削除された場合でも、プログラムはTNSファイルで引き続き使用できます。

## Pythonエディタでの作業

**[menu]**を押すと、ドキュメントツールメニューが表示されます。これらのメニューオプションを使うと、プログラムのコードブロックを追加、移動、コピーできます。

### ドキュメントツールメニュー

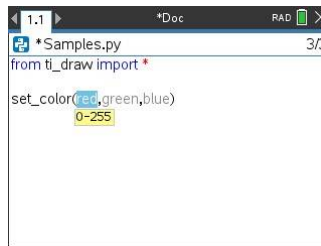


モジュールメニューから選択した項目は、関数の各引数のインラインプロンプト(行内入力要請)を持つコードテンプレートをエディタに自動的に追加します。**[tab]**(進む)または**[ctrl+tab]**(戻る)を押すと、ある引数から次の引数に移動できます。ツールチップやポップアップリストが利用可能なとき表示されます。これは、必要な値を選択するのに役立ちます。

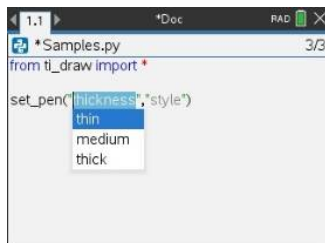
### インラインプロンプト



### ツールチップ

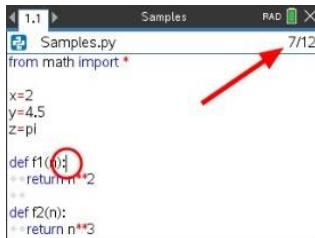


### ポップアップリスト



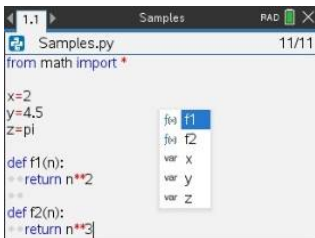


プログラム名の右側の数字は、現在カーソルがある行の番号とプログラムの全行数を示しています。



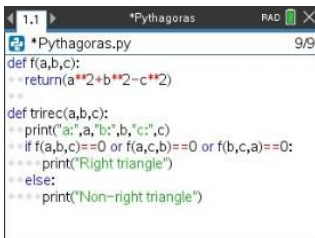
```
1.1 Samples.py 7/12
from math import *
x=2
y=4.5
z=pi
def f1(n):
    return n**2
def f2(n):
    return n**3
```

現在のカーソル位置の上の行で定義されたグローバル関数と変数は、`var`を押してリストから選択することができます。



```
1.1 Samples.py 11/11
from math import *
x=2
y=4.5
z=pi
def f1(n):
    return n**2
def f2(n):
    return n**3
```

プログラムにコードを追加すると、エディタはキーワード、演算子、コメント、文字列、インデント(字下げ、薄い灰色のひし形◆)を種々の色で表示して要素を識別しやすくします。



```
1.1 *Pythagoras.py 9/9
def f(a,b,c):
    return(a**2+b**2-c**2)
def trirec(a,b,c):
    print("a",a,"b",b,"c",c)
    if f(a,b,c)==0 or f(a,c,b)==0 or f(b,c,a)==0:
        print("Right triangle")
    else:
        print("Non-right triangle")
```

## プログラムの保存と実行

プログラムが終了したら、`menu`を押して`Run > Check Syntax & Save` (実行>構文の確認と保存)を選択します。これにより、Pythonプログラムの構文がチェックされ、TNSファイルに保存されます。

**Note:** プログラムに変更があり未保存のときは、プログラム名の横にアスタリスク(\*)が表示されます。



```
1.1 *Pythagoras.py 9/9
```

プログラムを実行するには、`menu`を押して`Run > Run` (実行>実行)を選択します(あるいはショートカットキーで、`ctrl+R`と押します)。これにより、現在のプログラムが次ページのPythonシェルページで実行されます。次ページがシェルでない場合は新規プログラムが新たに開いたPythonシェルページで実行されます。

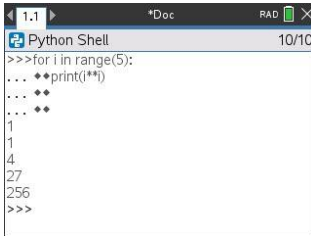
**Note:** プログラムを実行すると、構文が自動的にチェックされ、プログラムは保存されます。

# Pythonシェル

PythonシェルはPythonプログラム、他のPythonコード、または単純なコマンドを実行するインタプリタ(interpreter)です。

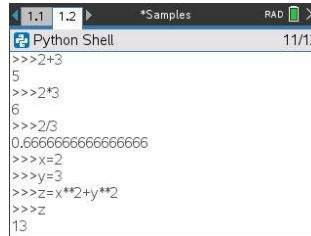
**Note:** interpreterは「通訳者」の意味で、コンピュータでプログラムを実行する方法の1つです。プログラムを1命令ずつ変換して実行するため、動きを確認しながらコードを記述したり、エラーが発生した個所をすぐに特定したりできます。

## Pythonコード



```
>>>for i in range(5):
...     **print(i**i)
...     **
...     **
1
1
4
27
256
>>>
```

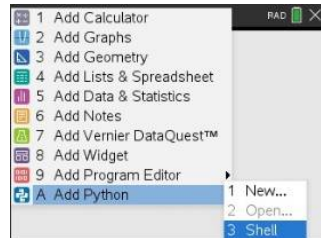
## 簡単なコマンド



```
>>>2+3
5
>>>2*3
6
>>>2/3
0.6666666666666666
>>>x=2
>>>y=3
>>>z=x**2+y**2
>>>z
13
```

## Pythonシェルページの追加

現在のプロブレムに新規のPythonシェルページを追加するには、**menu**を押してAdd Python > Shell (Pythonを追加>シェル)を選択します。



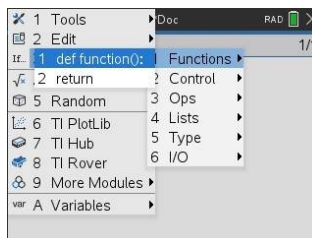
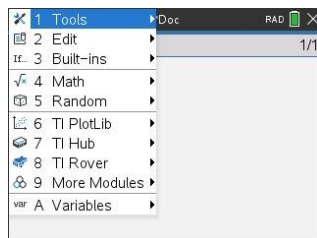
Pythonシェルは**menu**を押してRun > Run (実行>実行)を選択しプログラムを実行することにより、Pythonエディタから起動することもできます。



## Pythonシェルでの作業

**menu**を押すと、ドキュメントツールメニューが表示されます。これらのメニューオプションを使うと、コードのブロックを追加、移動、コピーができます。

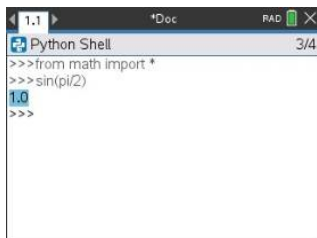
### ドキュメントツールメニュー



**Note:** 使用可能なモジュールからメソッドを使う場合、Pythonコーディング環境同様、必ず最初に**import module**ステートメントを実行してください。

シェル出力の利用は、以前の入力や出力を選択してコピーできる電卓アプリに似ています。シェル、エディタ、他のアプリの場所で使うことができます。

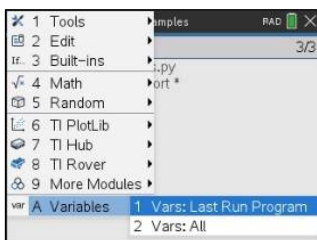
上矢印で選択し、コピーして目的の場所に貼り付けます。



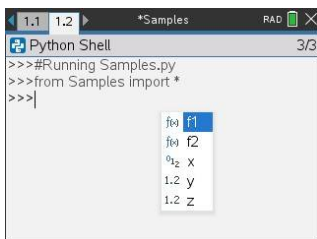
最後に実行したプログラムのグローバル関数と変数は、**var** または **ctrl+L** を押してリストから選択するか、**menu** を押して **Variables > Vars: Last Run Program** (変数>変数: 最後の実行プログラム) を選択することで挿入できます。

最後に実行したプログラムとインポートしたモジュールのグローバル関数と変数のリストから選択するには、**menu** を押して、**Variables > Vars: All** (変数>変数: すべて) を選択します。

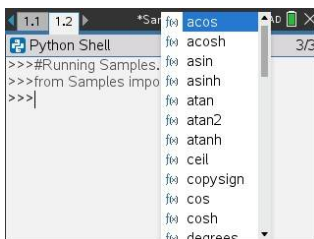
#### 変数メニュー



#### 最終実行プログラム変数



#### すべての変数



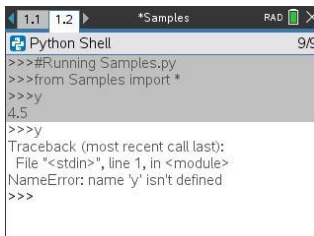
同じプロブレムのPythonシェルページは、すべて同じ状態(ユーザー定義とインポートされた変数定義)を共有します。そのプロブレムでPythonプログラムを保存または実行するか、**[menu]**を押して**Tools > Reinitialize Shell** (ツール>シェルの再初期化)を選択すると、シェル履歴の背景が灰色になり、前の状態が無効になったことを示します。

保存または再初期化する前



```
Python Shell 5/5
>>>#Running Samples.py
>>>from Samples import *
>>>y
4.5
>>>
```

保存または再初期化後



```
Python Shell 9/9
>>>#Running Samples.py
>>>from Samples import *
>>>y
4.5
>>>y
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'y' isn't defined
>>>
```

**Note:** **[menu] Tools > Clear History**(ツール>履歴のクリア)オプションは、シェル内の以前の入力や出力をクリアしますが、変数は引き続き使えます。

## メッセージ

Pythonセッション中に、エラーやその他の情報メッセージが表示される場合があります。プログラムの実行時にシェルにエラーが表示されると、プログラムの行番号が表示されます。

**[ctrl][menu]**を押して**Go to Python Editor** (Pythonエディタに移動)を選択します。エディタで**[menu]**を押して、**Edit > Go to Line** (編集>行に移動)を選択します。行番号を入力して**[enter]**を押します。エラーが発生した行の最初の文字にカーソルが表示されます。

## 実行中のプログラムの中断

プログラムまたは関数の実行中は、ビジーポインタが表示されます。

▶プログラムまたは機能を停止するには、

- Windows® : F12キーを押します。
- Mac® : F5キーを押します。
- グラフ電卓 : **[on]**を押します。

### TI-Nspireのドキュメント

TI-Nspireはコンピュータと同じファイル構造、すなわちフォルダーがあって、その中にドキュメントを作成・保存するという構造です。ドキュメントは複数のプロブレム(Problem)からなり、各プロブレムには複数のページを作成することができます。1ページが1画面にあたります。1つのドキュメントには最大30プロブレム、1つのプロブレムには最大50ページ作成できます。定義された変数や関数は、作成されたプロブレム内のみ有効です。

# Pythonメニュー

このセクションでは、PythonエディタとPythonシェルのすべてのメニューとその項目、そしてそれぞれの簡単な説明を一覧表示します。

**Note:** キーボードショートカットのあるメニュー項目の場合、Mac®ユーザーはCtrlを使用する場所で⌘ (Cmd)に置き換える必要があります。TI-Nspire™ハンドヘルドおよびソフトウェアショートカットの完全なリストについては、TI-Nspire™テクノロジーeGuideを参照してください。

---

Actions(アクション)メニュー .....	14
Run(実行)メニュー .....	14
Tools(ツール)メニュー .....	14
Edit(編集)メニュー .....	15
Built-ins(組み込み)メニュー .....	16
Math(数学)メニュー .....	18
Random(乱数)メニュー .....	20
TI PlotLib(TI プロットライブラリ)メニュー .....	20
TI Hub(TI ハブ)メニュー .....	22
TI Rover(TI ローバー)メニュー .....	30
Complex Math(複素数)メニュー .....	36
Time(時間)メニュー .....	36
TI System(TI システム)メニュー .....	37
TI Draw(TI 描画)メニュー .....	38
TI Image(TI 画像)メニュー .....	40
Variables(変数)メニュー .....	42

---

## Actions(アクション)メニュー

**Note:** これはエディタにのみ適用されます。

項目	説明
New	名前を入力し、新規プログラムのタイプを選択する <b>New(新規)</b> ダイアログボックスを開きます。
Open	現在のドキュメントで利用可能なプログラムのリストを開きます。
Create Copy	現在のプログラムを別の名前で保存できる <b>Create Copy(コピーを作成)</b> ダイアログボックスを開きます。
Rename	現在のプログラムの名前を変更する <b>Rename(名前の変更)</b> ダイアログボックスを開きます。
Close	現在のプログラムを閉じます。
Settings	エディタとシェルの両方のフォントサイズを変更できる <b>Settings(設定)</b> ダイアログボックスを開きます。
Install as Python module	現在のTNSファイルのPython構文をチェックし、それをPyLibフォルダーに移動します。

## Run(実行)メニュー

**Note:** これはエディタにのみ適用されます。

項目	ショートカット	説明
Run	Ctrl+R	構文をチェックし、プログラムを保存して、Pythonシェルで実行します。
Check Syntax & Save	Ctrl+B	構文をチェックし、プログラムを保存します。
Go to Shell	なし	現在のプログラムに関連するシェルにフォーカスを移動するか、エディタの横に新しいシェルページを開きます。

## Tools(ツール)メニュー

**Note:** これはシェルにのみ適用されます。

項目	ショートカット	説明
Rerun Last Program	Ctrl+R	現在のシェルに関連する最後のプログラムを再実行します。

Go to Python Editor	なし	現在のシェルに関連するエディタページを開きます。
Run	なし	現在のドキュメントで利用可能なプログラムのリストを開きます。 選択後、選択したプログラムが実行されます。
Clear History	なし	現在のシェルの履歴をクリアしますが、シェルを再初期化しません。
Reinitialize Shell	なし	現在のプロブレムで開いているすべてのシェルページの状態をリセットします。 定義されたすべての変数とインポートされた関数は使用できません。
dir()	なし	インポートステートメントの後に使用すると、指定したモジュールの関数のリストが表示されます。
From PROGRAM import *	なし	現在のドキュメントで利用可能なプログラムのリストを開きます。 選択後、インポートステートメントがシェルに貼り付けられます。
Install as Python Module	なし	バイナリ形式のモジュールに対してのみ有効になります。現在のTNSファイルをPyLibフォルダーに移動します。

## Edit(編集)メニュー

**Note:** Ctrl+Aは、コードや出力のすべての行を選択します。これは、切り取りまたは削除(エディタのみ)、またはコピーと貼り付け(エディタとシェル)のとき使います。

項目	ショートカット	説明
Indent	TAB*	現在の行または選択した行のテキストをインデント(字下げ)します。 *不完全なインラインプロンプト(行内入力要請)があるとき、TABは次のプロンプト(入力要請)に移動します。
Dedent	Shift+TAB**	現在の行または選択した行のインデント(字下げ)を解除します(デインデント)。 **不完全なインラインプロンプト(行内入力要請)があるとき、Shift+TABを押すと前のプロンプト(入力要請)に移動します。
Comment/Uncomment	Ctrl+T	現在の行の先頭にコメント記号(#)を追加/削除します。

Insert Multi-line String	なし	(エディタのみ)複数行の文字列テンプレートを挿入します。
Find	Ctrl+F	(エディタのみ) <b>Find</b> (検索)ダイアログボックスを開き、現在のプログラムで入力された文字列を検索します。
Replace	Ctrl+H	(エディタのみ) <b>Replace</b> (置換)ダイアログボックスを開き、現在のプログラムで入力された文字列を検索します。
Go to Line	Ctrl+G	(エディタのみ) <b>Go to Line</b> (行に移動)ダイアログボックスを開き、現在のプログラムの指定された行にジャンプします。
Beginning of Line	Ctrl+8	カーソルを現在の行の先頭に移動します。
End of Line	Ctrl+2	カーソルを現在の行の末尾に移動します。
Jump to Top	Ctrl+7	カーソルをプログラムの最初の行の先頭に移動します。
Jump to Bottom	Ctrl+1	カーソルをプログラムの最後の行の末尾に移動します。

## Built-ins(組み込み)メニュー

### Functions(関数)

項目	説明
def function():	指定された変数の関数を定義します。
return	関数によって生成された値を定義します。

### Control(制御)

項目	説明
if..	条件付きステートメント
if..else..	条件付きステートメント
if..elif..else..	条件付きステートメント
for index in range(size):	ある範囲で繰り返します。
for index in range(start,stop):	ある範囲で繰り返します。
for index in range (start,stop,step):	ある範囲で繰り返します。



for index in list:	リストの要素を繰り返し処理します。
while..	条件がFalseと評価されるまで、コードブロック内のステートメントを実行します。
elif:	条件付きステートメント
else:	条件付きステートメント

## Ops(演算子)

項目	説明
x=y	変数の値を設定します。
x==y	イコール(==)比較演算子を貼り付けます。
x!=y	ノットイコール(!=)比較演算子を貼り付けます。
x>y	より大きい(>)比較演算子を貼り付けます。
x>=y	以上(>=)比較演算子を貼り付けます
x<y	より小さい(<)比較演算子を貼り付けます。
x<=y	以下(<=)比較演算子を貼り付けます。
and	かつ(and)論理演算子を貼り付けます。
or	または(or)論理演算子を貼り付けます。
not	否定(not)論理演算子を貼り付けます。
True	Trueブール値を貼り付けます。
False	Falseブール値を貼り付けます。

## Lists(リスト)

項目	説明
[]	ブラケット([])を貼り付けます。
list()	数列をリスト形式に変換します。
len()	リストの要素数を返します。lenはlength(長さ)の省略形。
max()	リストの最大値を返します。
min()	リストの最小値を返します。
.append()	このメソッドは要素をリストに追加します。

<code>.remove()</code>	このメソッドは要素の最初のインスタンスをリストから削除します。 (インスタンス：Pythonには文字型や数値型など様々な型があり、それらの型の種類のことをクラスといいます。プログラムで実際に使うには型を実体化します。型の実体のことをインスタンスといいます。)
<code>range(start,stop,step)</code>	数値のセットを返します。
<code>for index in range(start,stop,step)</code>	範囲を反復するために使用されます。
<code>.insert()</code>	このメソッドは指定された位置に要素を追加します。
<code>.split()</code>	このメソッドは指定された区切り文字で区切られた要素を持つリストを返します。
<code>sum()</code>	リストの要素の合計を返します。
<code>sorted()</code>	ソートされたリストを返します。
<code>.sort()</code>	このメソッドはリストをその場でソートします。

## Type(タイプ)

項目	説明
<code>int()</code>	整数部分を返します。
<code>float()</code>	浮動小数点の値を返します。
<code>round(x,ndigits)</code>	指定された桁数に四捨五入した浮動小数点の数を返します。
<code>str()</code>	文字列を返します。
<code>complex()</code>	複素数を返します。
<code>type()</code>	オブジェクトのタイプを返します。

## I/O(入出力)

項目	説明
<code>print()</code>	引数を文字列として表示します。
<code>input()</code>	ユーザーに入力を求めます。
<code>eval()</code>	文字列として表された式を評価します。
<code>.format()</code>	このメソッドは指定された文字列をフォーマットします。

## Math(数学)メニュー

**Note:** このモジュールを使う新規プログラムを作成するときは、**Math Calculations**(数学計算)プログラムタイプを使うことをお勧めします。これにより、関連するすべてのモジュールが確実にインポートされます。

項目	説明
<code>from math import *</code>	mathモジュールからすべてのメソッド(関数)をインポートします。
<code>fabs()</code>	実数の絶対値を返します。
<code>sqrt()</code>	実数の平方根を返します。
<code>exp()</code>	$e^{**x}$ を返します。
<code>pow(x,y)</code>	$x^y$ を返します。
<code>log(x,base)</code>	$\log_{base}(x)$ を返します。底のない $\log(x)$ は自然対数 $x$ を返します。
<code>fmod(x,y)</code>	$x$ と $y$ のモジュール値を返します。 $x$ と $y$ がfloatの場合に使用します。
<code>ceil()</code>	実数以上の最小の整数を返します。
<code>floor()</code>	実数以下の最大の整数を返します。
<code>trunc()</code>	実数を整数に切り捨てます。
<code>frexp()</code>	組 $(y, n)$ を返します。ここで、 $x=y*2^{**n}$ です。

## Const(定数)

項目	説明
<code>e</code>	定数 $e$ の値を返します。
<code>pi</code>	定数 $\pi$ の値を返します。

## Trig(三角関数)

項目	説明
<code>radians()</code>	度単位の角度をラジアンに変換します。
<code>degrees()</code>	ラジアン単位の角度を度に変換します。
<code>sin()</code>	引数の $\sin$ (正弦)をラジアンで返します。
<code>cos()</code>	引数の $\cos$ (余弦)をラジアンで返します。
<code>tan()</code>	引数の $\tan$ (正接)をラジアンで返します。
<code>asin()</code>	引数のアークサイン(逆正弦)をラジアンで返します。
<code>acos()</code>	引数のアークコサイン(逆余弦)をラジアンで返します。

atan()	引数のアークタンジェント(逆正接)をラジアンで返します。
atan2(y,x)	y/xのアークタンジェント(逆正接)をラジアンで返します。

## Random(乱数)メニュー

**Note:** このモジュールを使う新規プログラムを作成するときは、**Random Simulations**(乱数シミュレーション)プログラムタイプを使用することをお勧めします。これにより、関連するすべてのモジュールが確実にインポートされます。

項目	説明
from random import *	randomモジュールからすべてのメソッドをインポートします。
random()	0から1.0までの浮動小数点数を返します。
uniform(min,max)	min<=x<=maxとなるような乱数x(float)を返します。
randint(min,max)	最小値と最大値の間のランダムな整数を返します。
choice(sequence)	空でない数列からランダムな要素を返します。
randrange(start,stop,step)	開始から停止までstep値ごと乱数を返します。
seed()	乱数生成を初期化します。

## TI PlotLib(TI プロットライブラリ)メニュー

**Note:** このモジュールを使う新規プログラムを作成するときは、**Plotting (x,y) & Text**プログラムタイプを使用することをお勧めします。これにより、関連するすべてのモジュールが確実にインポートされます。

項目	説明
import ti_plotlib as plt	"plt"名前空間のti_plotlibモジュールからすべてのメソッド(関数)をインポートします。その結果、メニューから貼り付けられた関数名の前にすべて"plt."が付きます。

## Setup (設定)

項目	説明
cls()	プロットキャンバスをクリアします。
grid(x-scale,y-scale,"style")	x軸とy軸に指定されたスケールを使ってグリッド(格子)を表示します。

<code>window(xmin,xmax,ymin,ymax)</code>	指定された水平間隔(xmin, xmax)と垂直間隔(ymin, ymax)を割り当てられたプロット領域(ピクセル)に対応させることにより、プロットウィンドウを定義します。
<code>auto_window(x-list,y-list)</code>	<code>auto_window()</code> の前にプログラムで指定されたx-listとy-list内のデータ範囲に合うように、プロットウィンドウを自動スケールリングします。
<code>axes("mode")</code>	プロット領域の指定されたウィンドウに軸を表示します。
<code>labels("x-label","y-label",x,y)</code>	行位置x, yのプロット軸に"x-label"と"y-label"ラベルを表示します。
<code>title("title")</code>	ウィンドウの最上行中央に"title"を表示します。
<code>show_plot()</code>	バッファリング(緩衝)された図面出力を表示します。 <code>use_buffer()</code> と <code>show_plot()</code> 関数は、画面に複数のオブジェクトを表示すると遅延が発生する可能性がある場合に役立ちます(ほとんどの場合は必要ありません)。
<code>use_buffer()</code>	オフスクリーンバッファ(off-screen buffer)を有効にして、描画を高速化します。

## Draw (描画)

項目	説明
<code>color(red,green,blue)</code>	以降のすべてのグラフ/プロットの色を設定します。
<code>cls()</code>	プロットキャンバスをクリアします。
<code>show_plot()</code>	プログラムで設定したプロットの表示を実行します。
<code>scatter(x-list,y-list,"mark")</code>	指定されたマークスタイルで、(x-list, y-list)から順序対の数列をプロットします。
<code>plot(x-list,y-list,"mark")</code>	指定されたx-listとy-listからの順序対を使って線をプロットします。
<code>plot(x,y,"mark")</code>	指定されたマークスタイルで、x座標とy座標を使って点をプロットします。
<code>line(x1,y1,x2,y2,"mode")</code>	(x1,y1)から(x2,y2)までの線分をプロットします。
<code>lin_reg(x-list,y-list,"display")</code>	x-list, y-listの線形回帰モデル $ax + b$ を計算して描画します。
<code>pen("size","style")</code>	次の <code>pen()</code> が実行されるまで、後続のすべての行の外観を設定します。
<code>text_at(row,"text","align")</code>	指定された"align"(位置合わせ)のプロット領域に"text"を表示します。

## Properties (プロパティ)

項目	説明
xmin	plt.xminとして定義されたウィンドウ引数に指定された変数。
xmax	plt.xmaxとして定義されたウィンドウ引数に指定された変数。
ymin	plt.yminとして定義されたウィンドウ引数に指定された変数。
ymax	plt.ymaxとして定義されたウィンドウ引数に指定された変数。
m	プログラムでplt.linreg()が実行された後、傾きmと切片bの計算値はplt.mとplt.bに格納されます。
b	プログラムでplt.linreg()が実行された後、傾きaと切片bの計算値はplt.aとplt.bに格納されます。

## TI Hub(TI ハブ)メニュー

**Note:** このモジュールを使う新規プログラムを作成するときは、Hub Projectプログラムタイプを使用することをお勧めします。これにより、関連するすべてのモジュールが確実にインポートされます。

項目	説明
from ti_hub import *	ti_hubモジュールからすべてのメソッドをインポートします。

## Hub Built-in Devices > Color Output (ハブ内蔵デバイス>色出力)

項目	説明
rgb(red,green,blue)	RGB LEDの色を設定します。
blink(frequency,time)	選択した色の点滅頻度と持続時間を設定します。
off()	RGB LEDをオフにします。

## Hub Built-in Devices > Light Output (ハブ内蔵デバイス>光出力)

項目	説明
on()	LEDをオンにします。
off()	LEDをオフにします。
blink(frequency,time)	LEDの点滅頻度と持続時間を設定します。

## Hub Built-in Devices > Sound Output (ハブ内蔵デバイス>サウンド出力)

項目	説明
tone(frequency,time)	指定された周波数のトーンを指定された時間再生します。
note("note",time)	指定されたノートが指定された時間再生します。 ノートは、ノート名とオクターブを使って指定されます。 例 : A4, C5 ノート名は、C, CS, D, DS, E, F, FS, G, GS, A, AS, Bです。 オクターブ数の範囲は1~9です。
tone(frequency,time,tempo)	指定された時間とテンポで指定された周波数のトーンを再生します。 テンポは、0~10の範囲の1秒当たりのピープ音の数を定義します。
note("note",time,tempo)	指定された時間とテンポで指定されたノートを演奏します。 ノートは、ノート名とオクターブを使って指定されます。 例 : A4, C5 ノート名は、C, CS, D, DS, E, F, FS, G, GS, A, AS, Bです。 オクターブ数の範囲は1~9です。 テンポ番号の範囲は0~10です。

## Hub Built-in Devices > Brightness Input (ハブ内蔵デバイス>明るさ入力)

項目	説明
measurement()	内蔵のBRIGHTNESS(光レベル)センサを読み取り、読み取り値を返します。 既定値の範囲は0~100です。これは、range()関数を使って変更できます。
range(min,max)	光レベルセンサからの読み取り値をマッピングするための範囲を設定します。両方が欠落している場合、または値がNoneに設定されている場合は、既定値の明るさの範囲である0~100が設定されます。

## Add Input Device (入力デバイスの追加)

このメニューには、ti\_hubモジュールでサポートされているセンサ(入力デバイス)のリストがあります。すべてのメニュー項目はオブジェクト名を貼り付け、センサで使用される変数とポートが表示されます。各センサには、センサの値を返すmeasurement()メソッドがあります。

項目	説明
DHT (Digital Humidity & Temp)	現在の温度、湿度、センサタイプ、最後にキャッシュされた読み取りステータスから成るリストを返します。

Ranger	<p>指定された超音波レンジャー(ultrasonic ranger)からの現在の距離の測定値を返します。</p> <ul style="list-style-type: none"> <li>• <code>measurement_time()</code> - 超音波信号がオブジェクトに到達するのにかかる時間(飛行時間)を返します。</li> </ul>
Light Level	<p>外部の光(brightness)センサからの明るさレベルを返します。</p>
Temperature	<p>外部の温度センサからの温度の測定値を返します。</p> <p>既定値の設定では、IN 1, IN 2, IN 3ポートでSeed温度センサをサポートします。</p> <p>TI Innovator™Hubのブレッドボードパック(Breadboard Pack)のTI LM19温度センサを使うには、使用中のBBピンへのポートを編集し、オプションの引数"TIANALOG"を使います。</p> <p>例：<code>mylm19=temperature("BB 5", "TIANALOG")</code></p>
Moisture	<p>湿度センサの読み取り値を返します。</p>
Magnetic	<p>磁場の存在を検出します。</p> <p>フィールドの存在を判別するしきい値(threshold value)は、<code>trigger()</code>関数を介して設定されます。</p> <p>しきい値の既定値は150です。</p>
Vernier	<p>コマンドで指定されたVernierアナログセンサから値を読み取ります。次のVernierセンサをサポートしています。</p> <ul style="list-style-type: none"> <li>• <code>temperature</code> (温度) - ステンレススチール温度センサ</li> <li>• <code>lightlevel</code> (光) - TI光レベルセンサ</li> <li>• <code>pressure</code> (圧力) - 旧ガス圧力センサ</li> <li>• <code>pressure</code> (圧力) - 新しいガス圧力センサ</li> <li>• <code>pH</code> (pH) - pHセンサ</li> <li>• <code>force10</code> (力) - ±10N設定, デュアルフォースセンサ</li> <li>• <code>force50</code> (力) - ±50N設定, デュアルフォースセンサ</li> <li>• <code>accelerometer</code> (加速度) - Low-G(低G)加速度計</li> <li>• <code>generic</code> (ジェネリック, 汎用) - 上記でサポートされていない他のセンサの設定や方程式の係数を設定する<code>calibrate()</code> APIの使用を許可します。</li> </ul>
Analog In	<p>アナログ入力ジェネリックデバイスの使用をサポートします。</p>
Digital In	<p>DIGITALオブジェクトに接続されているデジタルピンの現在の状態、またはオブジェクトに最後に設定されたデジタル出力値のキャッシュされた状態を返します。</p>
Potentiometer	<p>ポテンショメータ(可変抵抗器)センサをサポートします。</p> <p>センサ範囲は<code>range()</code>関数で変更できます</p>



項目	説明
Thermistor	<p>サーミスタセンサ(温度変化により抵抗値が変化するセンサ)を読み取ります。</p> <p>既定値の係数は、10KΩの固定抵抗とともに使った場合、TI-Innovator™Hubのブレッドボードバック(Breadboard Pack)に含まれているサーミスタと一致するように設計されています。</p> <p>サーミスタの新しいキャリブレーション係数と基準抵抗のセットは、<code>calibrate()</code>関数を使って設定できます。</p>
Loudness	音センサ(sound loudness sensor)をサポートします。
Color Input	<p>I2C接続のカラー入力センサへのインターフェースを提供します。</p> <p>I2Cポートに加えて<code>bb_port</code>ピンを使って、カラーセンサのLEDを制御します。</p> <ul style="list-style-type: none"> <li>• <code>color_number()</code>: センサが検出している色を表す1~9の値を返します。数字は次のマッピングごとの色を表しています: <ul style="list-style-type: none"> <li>1: Red (赤)</li> <li>2: Green (緑)</li> <li>3: Blue (青)</li> <li>4: Cyan (シアン, 青緑色)</li> <li>5: Magenta (マゼンタ, 赤紫色)</li> <li>6: Yellow (黄)</li> <li>7: Black (黒)</li> <li>8: White (白)</li> <li>9: Gray (グレー)</li> </ul> </li> <li>• <code>red()</code>: 検出されている赤の色レベル強度を表す0~255の値を返します。</li> <li>• <code>green()</code>: 検出されている緑の色レベル強度を表す0~255の値を返します。</li> <li>• <code>blue()</code>: 検出されている青の色レベル強度を表す0~255の値を返します。</li> <li>• <code>gray()</code>: 検出されているグレーレベルを表す0~255の値を返します。ここで、0は黒、255は白です。</li> </ul>
BB Port	<p>10個のBBポートピンすべてをデジタル入力/出力ポートの組み合わせとして使うサポートをします。</p> <p>初期化関数には、10ピンのサブセットの使用を可能にするオプションの"mask"パラメータがあります。</p> <ul style="list-style-type: none"> <li>• <code>read_port()</code>: BBポートの入力ピンの電流値を読み取ります。</li> <li>• <code>write_port(value)</code>: 出力ピンの値を指定された値に設定します。値は0~1023です。マスクが指定されている場合、値は<code>var=bbport(mask)</code>操作のマスク値に対しても調整されることに注意します。</li> </ul>

Hub Time	内蔵のミリ秒タイマーへのアクセスを提供します。
TI-RGB Array	<p>TI-RGB Arrayをプログラミングするための関数を提供します。初期化機能はオプションの"LAMP"パラメータを受け入れ、外部電源を必要とするTI-RGB Arrayの高輝度モードを有効にします。</p> <ul style="list-style-type: none"> <li>• <b>set(led_position, r,g,b):</b> 特定のled_position(0~15)を指定されたr, g, b値に設定します。ここで, r, g, bは0~255の値です。</li> <li>• <b>set(led_list,red,green,blue):</b> "led_list"で定義されたLEDを"red", "green", "blue"で指定された色に設定します。"led_list"は0から15までのLEDのインデックスを含むPythonリストです。たとえば, set([0,2,4,6,15], 0, 0, 255)はLED 0, 2, 4, 6, 15を青色に設定します。</li> <li>• <b>set_all(r,g,b):</b> Array内のすべてのRGB LEDを同じr, g, b値に設定します。</li> <li>• <b>all_off():</b> Array内のすべてのRGBをオフにします。</li> <li>• <b>measurement():</b> RGB ArrayがTI-Innovator™から使っているおおよその電流引き込みをミリアンペアで返します。</li> <li>• <b>pattern(pattern):</b> 引数の値を0~65535の範囲のバイナリ値として使うと、表現の1の値が存在するピクセルがオンになります。LEDは255のpwmレベル値でREDとしてオンになります。</li> <li>• <b>pattern(value,red,green,blue):</b> "pattern"で定義されたLEDを"red", "green", "blue"で指定された色に設定します。</li> </ul>

## Add Output Device (出力デバイスの追加)

このメニューには、ti\_hubモジュールでサポートされている出力デバイスのリストがあります。すべてのメニュー項目はオブジェクトの名前を貼り付け、デバイスで使われる変数とポートを表示します。

項目	説明
LED	外部接続されたLEDを制御するための関数
RGB	外部RGB LEDの制御のサポート
TI-RGB Array	TI-RGB Arrayをプログラミングするための関数を提供します。
Speaker	TI-Innovator™ Hubで外部スピーカーをサポートするための関数 関数は上記の"sound"と同じです。
Power	TI-Innovator™ Hubで外部電源を制御するための関数 <ul style="list-style-type: none"> <li>• <b>set(value):</b> 電力レベルを指定された値(0~100)に設定します。</li> <li>• <b>on():</b> 電力レベルを100に設定します。</li> <li>• <b>off():</b> 電力レベルを0に設定します。</li> </ul>

Continuous Servo	<p>連続サーボモータを制御する関数</p> <ul style="list-style-type: none"> <li>• <b>set_cw(speed,time):</b> サーボは、指定された速度(0~255)で秒単位の特定の時間、時計回りに回転します。</li> <li>• <b>set_ccw(speed,time):</b> サーボは、指定された速度(0~255)で秒単位の特定の時間、反時計回りに回転します。</li> <li>• <b>stop():</b> 連続サーボを停止します</li> </ul>
Analog Out	アナログ入力ジェネリック(汎用)デバイスを使うための関数
Vibration Motor	<p>振動モーターを制御するための関数</p> <ul style="list-style-type: none"> <li>• <b>set(val):</b> 振動モーターの強度を"val" (0-255)に設定します。</li> <li>• <b>off():</b> 振動モーターをオフにします。</li> <li>• <b>on():</b> 振動モーターを最高レベルでオンにします。</li> </ul>
Relay	<p>リレーを制御するためのインターフェースを制御します。</p> <ul style="list-style-type: none"> <li>• <b>on():</b> リレーをオン状態に設定します。</li> <li>• <b>off():</b> リレーをオフ状態に設定します。</li> </ul>
Servo	<p>サーボモータを制御する関数</p> <ul style="list-style-type: none"> <li>• <b>set_position(pos):</b> スイープ(sweep)サーボ位置を-90~+90の範囲で設定します。</li> <li>• <b>zero():</b> スイープサーボをゼロ位置に設定します。</li> </ul>
Squarewave	<p>矩形波を生成するための関数</p> <p>参考: Square wave(矩形波「くけいは」)とは非正弦波形の基本的な一種で、電子工学や信号処理の分野で広く使われている波です。</p> <ul style="list-style-type: none"> <li>• <b>set(frequency,duty,time):</b> 既定値のデューティサイクルが50%(デューティが指定されていない場合)および"frequency(周波数)"で指定された出力周波数で出力矩形波を設定します。周波数は1~500Hzです。デューティサイクルは、指定されている場合、0~100%です。</li> </ul> <p>参考: デューティサイクルまたはデューティ比とは、周期的な現象において、"ある期間"に占める"その期間で現象が継続される期間"の割合です。制御、電気通信や電子工学で使います。</p> <ul style="list-style-type: none"> <li>• <b>off():</b> 矩形波をオフにします。</li> </ul>
Digital Out	<p>デジタル出力を制御するインターフェース</p> <ul style="list-style-type: none"> <li>• <b>set(val):</b> デジタル出力を"val"で指定された値(0または1)に設定します。</li> <li>• <b>on():</b> デジタル出力の状態を高(1)に設定します。</li> <li>• <b>off():</b> デジタル出力の状態を低(0)に設定します。</li> </ul>
BB Port	TI-RGB Arrayをプログラミングするための関数を提供します。上記の詳細を参照してください。

## Commands (命令)

項目	説明
<code>sleep(seconds)</code>	指定された秒数の間、プログラムを一時停止します。'time'モジュールからインポートされます。
<code>text_at(row,"text","align")</code>	指定された"align"(位置合わせ)のプロット領域に、指定された"text"を表示します。ti_plotlibモジュールの一部。
<code>cls()</code>	プロット用のシェル画面をクリアします。ti_plotlibモジュールの一部。
<code>while get_key() != "esc":</code>	"esc"キーが押されるまで、"while"ループでコマンドを実行します。
<code>get_key()</code>	押されたキーを表す文字列を返します。 '1'キーは"1"を返し、'esc'は"esc"を返します、など。 パラメータなし- <code>get_key()</code> -で呼び出されると、すぐに戻ります。 パラメータ- <code>get_key(1)</code> -を指定して呼び出されると、キーが押されるまで待機します。 ti_systemモジュールの一部。

## Ports (ポート)

以下は、TI-Innovator™ Hubで使用可能な入力ポートと出力ポートです。

項目
OUT 1
OUT 2
OUT 3
IN 1
IN 2
IN 3
BB 1
BB 2
BB 3
BB 4
BB 5
BB 6
BB 7
BB 8
BB 9
BB 10
I2C

## TI Rover(TI ローバー)メニュー

**Note:** このモジュールを使う新規プログラムを作成するときは、Rover Codingプログラムタイプを使用することをお勧めします。これにより、関連するすべてのモジュールが確実にインポートされます。

項目	説明
import ti_rover as rv	"rv"名前空間のti_roverモジュールからすべてのメソッド(関数)をインポートします。その結果、メニューから貼り付けられた関数名の前にすべて"rv."が付きます。

## Drive (ドライブ)

項目	説明
forward(distance)	Roverをグリッド単位で指定された距離だけ前方に移動します。
backward(distance)	Roverをグリッド単位で指定された距離だけ後方に移動します。
left(angle_degrees)	Roverを指定された角度(度単位)で左に回します。
right(angle_degrees)	Roverを指定された角度(度単位)で右に回します。
stop()	現在の動きを即座に停止します。
stop_clear()	現在の移動をただちに停止し、保留中のコマンドをすべてクリアします。
resume()	コマンド処理を再開します。
stay(time)	Roverは、指定された時間(秒単位)の間、所定の位置に留まります(オプション)。 時間が指定されていない場合、Roverは30秒間留まります。
to_xy(x,y)	Roverを仮想グリッド(virtual grid)上の座標位置(x, y)に移動します。
to_polar(r,theta_degrees)	Roverを仮想グリッド(virtual grid)上の極座標位置(r, theta)に移動します。 角度は度で指定されます。
to_angle(angle,"unit")	仮想グリッド(virtual grid)で指定された角度にRoverを回転させます。 角度は、仮想グリッドのx軸を指すゼロ角度を基準にしています。

## Drive > Drive with Options (ドライブ>オプション付きドライブ)

項目	説明
forward_time(time)	指定された時間, Roverを前方に移動します。
backward_time(time)	指定された時間, Roverを後方に移動します。
forward(distance,"unit")	指定された距離の間, 既定値の速度でRoverを前方に移動します。 距離は, グリッド単位, メートル, またはホイールの回転数で指定できます。
backward(distance,"unit")	指定された距離の間, 既定値の速度でRoverを後方に移動します。 距離は, グリッド単位, メートル, またはホイールの回転数で指定できます。
left(angle,"unit")	Roverを指定された角度だけ左に回します。 角度は, 度, ラジアン, gradian(グラード, 1グラードは直角(90度)の100分の1)で指定できます。
right(angle,"unit")	Roverを指定された角度だけ右に回します。 角度は, 度, ラジアン, gradian(グラード, 1グラードは直角(90度)の100分の1)で指定できます。
forward_time(time,speed,"rate")	指定された速度で, 指定された時間, Roverを前方に移動します。 速度は, グリッド単位/秒, メートル/秒, ホイール回転/秒で指定できます。
backward_time(time,speed,"rate")	指定された速度で, 指定された時間, Roverを後方に移動します。 速度は, グリッド単位/秒, メートル/秒, ホイール回転/秒で指定できます。
forward(distance,"unit",speed,"rate")	指定された速度で指定された距離だけRoverを前方に移動します。 距離は, グリッド単位, メートル, ホイールの回転数で指定できます。 速度は, グリッド単位/秒, メートル/秒, ホイール回転/秒で指定できます。
backward(distance,"unit",speed,"rate")	指定された速度で指定された距離だけRoverを後方に移動します。 距離は, グリッド単位, メートル, ホイールの回転数で指定できます。 速度は, グリッド単位/秒, メートル/秒, ホイール回転/秒で指定できます。

## Inputs (入力)

項目	説明
ranger_measurement()	Roverの前面にある距離センサ(ultrasonic distance sensor)を読み取り、現在の距離をメートル単位で返します。
color_measurement()	1から9までの値を返します。これは、Rover色入力センサによって感知される色を示します。 1 = red (赤) 2 = green (緑) 3 = blue (青) 4 = cyan (シアン, 青緑色) 5 = magenta (マゼンタ, 赤紫色) 6 = yellow (黄色) 7 = black (黒) 8 = grey (灰色) 9 = white (白)
red_measurement()	色入力センサによって感知された赤のレベルを示す0~255の値を返します。
green_measurement()	色入力センサによって感知された緑のレベルを示す0~255の値を返します。
blue_measurement()	色入力センサによって感知された青のレベルを示す0~255の値を返します。
gray_measurement()	色入力センサによって感知された灰色のレベルを示す0~255の値を返します。
encoders_gyro_measurement()	左右のホイールエンコーダカウンタと現在のジャイロ方位を含む値のリストを返します。
gyro_measurement()	ドリフトを含む現在のジャイロの読み取り値を度単位の値で返します。
ranger_time()	T1-Rover rangerからの超音波信号がオブジェクトに到達するまでにかかる時間("time of flight"「飛行時間」)を返します。

## Outputs (出力)

項目	説明
color_rgb(r,g,b)	Rover RGB LEDの色を、指定された赤、緑、青の値に設定します。
color_blink(frequency,time)	選択した色の点滅頻度と時間を設定します。
color_off()	Rover RGB LEDをオフにします。



motor_left(speed,time)	指定された時間、左側のモーター出力を指定された値に設定します。 速度は-255から255の範囲で、0が停止です。正の速度は反時計回りの回転で、負の速度は時計回りです。オプションの時間パラメータは、指定されている場合、0.05~655.35秒の有効範囲を持ちます。指定されていない場合、既定値5秒が使用されます。
motor_right(speed,time)	指定された時間、右側のモーター出力を指定された値に設定します。 速度は-255から255の範囲で、0が停止です。正の速度は反時計回りの回転で、負の速度は時計回りです。オプションの時間パラメータは、指定されている場合、0.05~655.35秒の有効範囲を持ちます。指定されていない場合、既定値5秒が使用されます。
motors("ldir",left_val,"rdir",right_val,time)	左右のホイールを指定された速度レベルに設定します。オプションの時間は秒単位です。 速度(left_val, right_val)の値は0から255の範囲で、0は停止です。ldirとrdirパラメータは、それぞれのホイールのCW(時計回り)またはCCW(反時計回り)回転を指定します。オプションの時間パラメータは、指定されている場合、0.05~655.35秒の有効範囲を持ちます。指定されていない場合、既定値5秒が使用されます。

## Path (経路)

項目	説明
waypoint_xythdrn()	x座標、y座標、時間、進行方向、移動距離、ホイールの回転数、現在の経路点のコマンド番号を読み取ります。これらすべての値を要素として含むリストを返します。  <b>参考：</b> ウェイポイント(waypoint)とは、経路上の地点情報をいいます。一般的には、緯度と経度の情報の組です。
waypoint_prev	x座標、y座標、時間、進行方向、移動距離、ホイールの回転数、前の経路点のコマンド番号を読み取ります。
waypoint_eta	経路点まで運転する推定時間を返します。
path_done()	Roverが移動しているか(0)、移動が終了しているか(1)に応じて0または1の値を返します。
pathlist_x()	最初から現在の経路点までのxの値のリストを返します。
pathlist_y()	最初から現在の経路点までのyの値のリストを返します。

pathlist_time()	最初から現在の経路点までの時間のリストを秒単位で返します。
pathlist_heading()	最初から現在の経路点までの進行方向のリストを返します。
pathlist_distance()	最初から現在の経路点までの移動距離のリストを返します。
pathlist_revs()	最初から現在の経路点までのホイールの回転数のリストを返します。
pathlist_cmdnum()	経路のコマンド番号のリストを返します。
waypoint_x()	現在の経路点のx座標を返します。
waypoint_y()	現在の経路点のy座標を返します。
waypoint_time()	以前の経路点から現在の経路点への移動に費やされた時間を返します。
waypoint_heading()	現在の経路点の(絶対)方位を返します。
waypoint_distance()	以前の経路点と現在の経路点の間の移動距離を返します。
waypoint_revs()	以前の経路点と現在の経路点の間を移動するのに必要な回転数を返します。

## Settings (設定)

項目	説明
units/s	1秒あたりのグリッド単位での速度のオプション
m/s	メートル/秒単位の速度のオプション
revs/s	1秒あたりのホイール回転数の速度のオプション
units	グリッド単位での距離のオプション
m	メートル単位の距離のオプション
revs	ホイール回転数の距離のオプション
degrees	度単位の方向を変えるオプション
radians	ラジアン単位の方向を変えるオプション
gradians	グラード(1グラードは直角(90度)の100分の1)単位の方向を変えるオプション
clockwise	ホイールの時計回りの方向を指定するオプション
counter-clockwise	ホイールの反時計回りの方向を指定するオプション

## Commands (命令)

以下のコマンドは、TI-Roverモジュールだけでなく他のモジュールから集めた機能です。

項目	説明
<code>sleep(seconds)</code>	指定された秒数の間、プログラムを一時停止します。time(時間)モジュールからインポートされます。
<code>text_at(row,"text","align")</code>	指定された"align"(位置合わせ)のプロット領域に"text"を表示します。ti_plotlibモジュールからインポートされます。
<code>cls()</code>	プロット用のシェル画面をクリアします。 ti_plotlibモジュールからインポートされます。
<code>while get_key() != "esc":</code>	"esc"キーが押されるまで、"while"ループでコマンドを実行します。
<code>wait_until_done()</code>	Roverが現在のコマンドを終了するまで、プログラムを一時停止します。 これは、Rover以外のコマンドをRoverのmotionと同期させるのに便利な方法です。
<code>while not path_done()</code>	Roverがすべての動きを終えるまで、"while"ループでコマンドを実行します。 path_done()関数は、Roverが移動しているか(0)、移動が終了しているか(1)に応じて、0または1の値を返します。
<code>position(x,y)</code>	仮想グリッド上のRover位置を指定されたx,y座標に設定します。
<code>position(x,y,heading,"unit")</code>	仮想グリッド(virtual grid)上のRoverの位置を指定されたx,y座標に設定し、進行方向が指定されている場合は、仮想x軸を基準にした仮想進行方向が設定されます(指定された角度の単位で)。0から360までの正の角度は正のx軸から反時計回り、0から-360までの負の角度は正のx軸から時計回りです。
<code>grid_origin()</code>	RVを(0,0)の現在のグリッド原点にあるように設定します。
<code>grid_m_unit(scale_value)</code>	仮想グリッド(virtual grid)間隔をメートル/単位(m/unit)で指定された値に設定します。0.1は既定値のm/unitであり、 1単位=100mmまたは10cmまたは1dmまたは0.1mに変換されます。 有効なscale_valueの範囲は0.01~10.0です。
<code>path_clear()</code>	既存の経路または経路点情報をすべてクリアします。
<code>zero_gyro()</code>	Roverジャイロを0.0の角度にリセットし、左右のホイールエンコーダーカウントをクリアします。

## Complex Math(複素数)メニュー

このサブメニューはMore Modules (その他のモジュール)の下にあります

項目	説明
<code>from cmath import *</code>	cmathモジュールからすべてのメソッドをインポートします。
<code>complex(real,imag)</code>	複素数を返します。
<code>rect(modulus,argument)</code>	極座標の複素数を直交座標に変換します。
<code>.real</code>	複素数の実部を返します。
<code>.imag</code>	複素数の虚部を返します。
<code>polar()</code>	直交座標の複素数を極座標に変換します。
<code>phase()</code>	複素数の偏角(位相)を返します。
<code>exp()</code>	$e^{**x}$ を返します。
<code>cos()</code>	複素数のcosを返します。
<code>sin()</code>	複素数のsinを返します。
<code>log()</code>	複素数の自然対数を返します。
<code>log10()</code>	複素数の、10を底とする対数を返します。
<code>sqrt()</code>	複素数の平方根を返します。

## Time(時間)メニュー

このサブメニューはMore Modules (その他のモジュール)の下にあります

項目	説明
<code>from time import *</code>	time(時間)モジュールからすべてのメソッドをインポートします。
<code>sleep(seconds)</code>	指定された秒数の間、プログラムを一時停止します。
<code>clock()</code>	現在のプロセッサ時間を秒単位の浮動小数点数として返します。
<code>localtime()</code>	2000年1月1日から秒で表される時間を、年、月、日、時、分、秒、平日、年日、夏時間(DST:Daylight Savings Time)フラグを含む9タプル(要素の追加や削除ができないリスト)に変換します。オプション(seconds)引数が指定されていない場合は、リアルタイムクロック(内蔵時計)が使用されます。
<code>ticks_cpu()</code>	任意のreference point(参照ポイント)を持つプロセッサ固有の、増加するミリ秒カウンタを返します。異なるシステム間で一貫して時間を測定するには、ticks_ms()を使用します。

ticks_diff()	ticks_cpu()またはticks_ms()への連続した呼び出し間の期間を測定します。この関数は、長期間を測定するのに使用しないでください。
--------------	--

## TI System(TI システム)メニュー

このサブメニューはMore Modules (その他のモジュール)の下にあります

**Note:** このモジュールを使う新規プログラムを作成するときは、Data Sharing(データ共有)プログラムタイプを使用することをお勧めします。これにより、関連するすべてのモジュールが確実にインポートされます。

項目	説明
from ti_system import *	ti_systemモジュールからすべてのメソッド(関数)をインポートします。
recall_value("name")	"name"という名前の事前に定義されたOS変数(値)を呼び出します。
store_value("name",value)	Python変数(値)を"name"という名前のOS変数に格納します。
recall_list("name")	"name"という名前の事前に定義されたOSリストを呼び出します。
store_list("name",list)	Pythonリスト(リスト)を"name"という名前のOSリスト変数に格納します。
eval_function("name",value)	事前に定義されたOS機能を指定された値で評価します。
get_platform()	ハンドヘルドは"hh"を返し、デスクトップは"dt"を返します。
get_key()	押されたキーを表す文字列を返します。 '1'キーは"1"を返し、'esc'は"esc"を返します、など。パラメータなしで呼び出されると(get_key()), すぐに戻ります。 パラメータを指定して呼び出されると(get_key(1)), キーが押されるまで待機します。
get_mouse()	マウスの座標を2要素のタプル(要素の追加や削除ができないリスト)として返します。キャンパスのピクセル位置、またはキャンパスの外側の場合は(-1,-1)のいずれかです。
while get_key() != "esc":	"esc"キーが押されるまで、"while"ループでコマンドを実行します。
clear_history()	シェルの履歴をクリアします。
get_time_ms()	ミリ秒単位の精度で時間をミリ秒単位で返します。 この機能は、実際の時刻を決定するのではなく、期間を計算するため使えます。

## TI Draw(TI 描画)メニュー

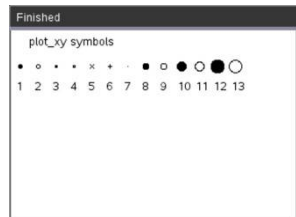
このサブメニューはMore Modules(その他のモジュール)の下にあります。

**Note:** このモジュールを使う新規プログラムを作成するときは、Geometry Graphics(幾何グラフィック)プログラムタイプを使用することをお勧めします。これにより、関連するすべてのモジュールが確実にインポートされます。

項目	説明
from ti_draw import *	ti_drawモジュールからすべてのメソッドをインポートします。

### Shape (形)

項目	説明
draw_line()	指定されたx1, y1座標からx2, y2までの線を描画します。
draw_rect()	指定された幅と高さで、指定されたx, y座標から始まる長方形を描画します。
fill_rect()	指定された幅と高さで、指定されたx, y座標で始まり、指定された色で塗りつぶされた長方形を描画します(定義されていない場合は、set_colorまたは黒を使用)。
draw_circle()	指定された半径で、指定されたx, y中心座標から始まる円を描画します。
fill_circle()	指定されたx, y中心座標から開始し、指定された半径で、指定された色で塗りつぶされた円を描画します(定義されていない場合は、set_colorまたは黒を使用)。
draw_text()	指定されたx, y座標で始まるテキスト文字列を描画します。
draw_arc()	指定された幅、高さ、角度で、指定されたx, y座標から始まる円弧を描画します。
fill_arc()	指定されたx, y座標で開始し、指定された幅、高さ、角度で、指定された色で塗りつぶされた円弧を描画します(定義されていない場合は、set_colorまたは黒を使用)。
draw_poly()	指定されたx-list, y-list値を使って多角形を描画します。
fill_poly()	指定されたx-list, y-list値を使って、指定された色で塗りつぶされた多角形を描画します(定義されていない場合は、set_colorまたは黒を使用)。
plot_xy()	指定されたx, y座標と、さまざまな形状と記号を表す1~13の指定番号を使って形状を描画します(右図)。



## Control (制御)

項目	説明
<code>clear()</code>	画面全体をクリアします。x, y, 幅, 高さパラメータとともに使って、既存の長方形をクリアできます。
<code>clear_rect()</code>	指定された幅と高さで、指定されたx, y座標の長方形をクリアします。
<code>set_color()</code>	別の色が設定されるまで、プログラムで続く形状の色を設定します。
<code>set_pen()</code>	図形を描画するときに指定した境界線の太さとスタイルを設定します(塗りつぶしコマンド(fill command)を使う場合は適用されません)。
<code>set_window()</code>	図形が描画されるウィンドウのサイズを設定します。 この関数は、データに一致するようにウィンドウのサイズを変更したり、描画キャンバスの原点(0, 0)を変更したりする場合に便利です。
<code>get_screen_dim()</code>	画面サイズのxmaxとymaxを返します。
<code>use_buffer()</code>	オフスクリーンバッファ(off-screen buffer)を有効にして、描画を高速化します。
<code>paint_buffer()</code>	バッファリング(緩衝)された図面出力を表示します。 <code>use_buffer()</code> と <code>paint_buffer()</code> 関数は、画面に複数のオブジェクトを表示すると遅延が発生する可能性がある場合に役立ちます。

### Notes :

- 既定値の設定では、画面の左上隅に原点(0, 0)があります。正のx軸は右を指し、正のy軸は下を指します。これは、`set_window()`関数を使って変更できます。
- `ti_draw`モジュールの関数は、グラフ電卓とデスクトップのグラフ電卓ビューでのみ使用できます。

## TI Image(TI 画像)メニュー

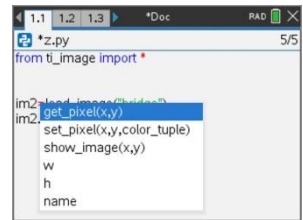
このサブメニューはMore Modules(その他のモジュール)の下にあります。

**Note:** このモジュールを使う新規プログラムを作成するときは、**Image Processing**(画像処理)プログラムタイプを使用することをお勧めします。これにより、関連するすべてのモジュールが確実にインポートされます。

項目	説明
<code>from ti_image import *</code>	ti_imageモジュールからすべてのメソッドをインポートします。
<code>new_image(width,height,(r,g,b))</code>	Pythonプログラムで使用するため、指定された幅と高さで新しい画像を作成します。 新しい画像の色は(r, g, b)の値によって定義されます。
<code>load_image("name")</code>	Pythonプログラムで使用するため、"name"で指定された画像をロードします。 画像は、NotesまたはGraphsアプリのいずれかで、TNSドキュメントの一部である必要があります。 "name"プロンプトには、画像名(以前に名前が付けられている場合)または挿入順序を示す番号が表示されます。
<code>copy_image(image)</code>	"image"変数で指定された画像のコピーを作成します。

## 画像オブジェクトのメソッド

画像オブジェクトに関連する追加の関数は、変数名の後にドット(.)を入力することによりエディタとシェルで使用できます。



- **get\_pixel(x,y):** (x, y)座標で定義された位置にあるピクセルの(r, g, b)値を取得します。

```
px_val = get_pixel(100,100)
print(px_val)
```

- **set\_pixel(x,y,color\_tuple):** 位置(x, y)のピクセルをcolor\_tupleで指定された色に設定します。

```
set_pixel(100,100, (0,0,255))
```

(100, 100)のピクセルを(0,0,255)カラーに設定します。

- **show\_image(x,y):** 画像を左上隅の位置(x, y)に表示します。
- **w, h, name:** 画像の幅、高さ、名前のパラメータを取得します。



## 例

```
from ti_image import *

# An image has been previously inserted into the TNS document in a
Notes application and named "bridge"
im1=load_image("bridge")
px_val = im1.get_pixel(100,100)
print(px_val)

# Set the pixel at 100,100 to blue (0,0,255)
im1.set_pixel(100,100,(0,0,255))
new_px = im1.get_pixel(100,100)
print(new_px)

# Print the width, height and name of the image
print(im1.w, im1.h, im1.name)
```

## Variables(変数)メニュー

**Note:** これらのリストには、他のTI-Nspire™アプリで定義されている変数は含まれていません。

項目	説明
Vars: Current Program	(エディタのみ) 現在のプログラムで定義されているグローバル関数と変数のリストを表示します。
Vars: Last Run Program	(シェルのみ) 最後に実行したプログラムで定義されたグローバル関数と変数のリストを表示します。
Vars: All	(シェルのみ) 最後に実行されたプログラムとインポートされたモジュールの両方からのグローバル関数と変数のリストを表示します。

# 付録

---

Pythonキーワード .....	44
Pythonキー対応 .....	45
例：Pythonプログラム .....	47


## Pythonキーワード

次のキーワードは、TI-Nspire™ Pythonに組み込まれています。

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

## Pythonキー対応

エディタやシェルにコードを入力するとき、キーボードは適切なPython操作を貼り付ける、あるいはメニューを開いて関数、キーワード、メソッド、演算子などを簡単に入力できるように設計されています。

キー	対応
	変数メニューを開く
	=記号を貼り付け
	カーソルの左側の文字を削除
	何もしない
	=記号を貼り付け
	選択した記号を貼り付け： <ul style="list-style-type: none"><li>• &gt;</li><li>• &lt;</li><li>• !=</li><li>• &gt;=</li><li>• &lt;=</li><li>• ==</li><li>• and</li><li>• or</li><li>• not</li><li>•  </li><li>• &amp;</li><li>• ~</li></ul>
	選択した関数を貼り付け： <ul style="list-style-type: none"><li>• sin</li><li>• cos</li><li>• tan</li><li>• atan2</li><li>• asin</li><li>• acos</li><li>• atan</li></ul>
	ヒントを表示
	:=を貼り付け
	**を貼り付け
	何もしない
	**2を貼り付け

キー	対応
$\sqrt{\quad}$	sqrt()を貼り付け
$\times$	かけ算記号(*)を貼り付け
""	二重引用符(")を1つ貼り付け
$\div$	わり算記号(/)を貼り付け
$\frac{\quad}{\quad}$	何もしない
$e^x$	exp()を貼り付け
$\ln$	log()を貼り付け
$10^x$	10**を貼り付け
$\log$	log(value,base)を貼り付け
(	(を貼り付け
)	)を貼り付け
[ ]	[ ]を貼り付け
{ }	{ }を貼り付け
(-)	引き算記号(-)を貼り付け
$\approx$	現在の行の後に新しい行を追加
EE	Eを貼り付け
? >	選択した記号を貼り付け： <ul style="list-style-type: none"> <li>• ?</li> <li>• !</li> <li>• \$</li> <li>• °</li> <li>• ' <ul style="list-style-type: none"> <li>• %</li> <li>• " <ul style="list-style-type: none"> <li>• :</li> <li>• ;</li> <li>• _</li> <li>• ¥</li> <li>• #</li> </ul> </li> </ul> </li> </ul>
$\pi$ >	"pi"を貼り付け
F	既存のフラグの動作
$\leftarrow$	現在の行の後に新しい行を追加

## 例：Pythonプログラム

以下のサンプルプログラムを使って、Pythonに慣れてください。これらは、**Examples**フォルダにある**Getting Started Python.tns**ファイルでも入手できます。

**Note:** インデント(字下げ)インジケータ(••)を含むサンプルコードをコピーしてTI-Nspire™に貼り付ける場合は、それらは実際のインデント◆◆に置き換える必要があります。

### Hello

```
# This program asks for your name and uses
# it in an output message.
# Run the program here by typing "Ctrl R"
```

```
name=input("What's your name? ")
print("Hello, ", name)
print("\n Press ctrl+R to run again")
```



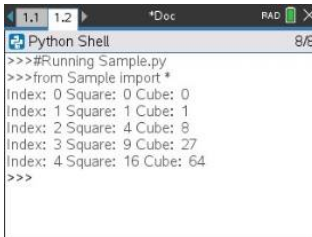
The screenshot shows a Python Shell window with the following text:

```
Python Shell 7/7
>>>#Running Sample.py
>>>from Sample import *
What's your name? Thomas
Hello, Thomas
>>>
 Press ctrl+R to run again
>>>
```

## Loop Example

```
# This program uses a "for" loop to calculate
# the squares and cubes of the first 5 numbers
# 0,1,2,3,4
# Note: Python starts counting at 0
```

```
for index in range(5):
    •square = index**2
    •cube = index**3
    •print("Index: ", index, "Square: ", square,
    •••"Cube: ", cube)
```



The screenshot shows a Python Shell window with the following output:

```
>>>#Running Sample.py
>>>from Sample import *
Index: 0 Square: 0 Cube: 0
Index: 1 Square: 1 Cube: 1
Index: 2 Square: 4 Cube: 8
Index: 3 Square: 9 Cube: 27
Index: 4 Square: 16 Cube: 64
>>>
```



## Heads or Tails

```
# Use random numbers to simulate a coin flip
# We will count the number of heads and tails
# Run the program here by typing "Ctrl R"

# Import all the functions of the "random" module
from random import *

# n is the number of times the die is rolled
def coin_flip(n):
    ...heads = tails = 0
    ...for i in range(n):
# Generate a random integer - 0 or 1
# "0" means head, "1" means tails
    ...side=randint(0,1)
    ...if (side == 0):
    .....heads = heads + 1
    ...else:
    .....tails = tails + 1
# Print the total number of heads and tails
    ...print(n, "coin flips: Heads: ", heads, "Tails: ", tails)

print("\nPress the Var key and select 'coin_flip()'")
print("In the ( ), enter a number of flips!")
```



The screenshot shows a Python Shell window titled "Python Shell" with a file path of "1.1 1.2" and a document name of "\*Doc". The shell displays the following text:

```
>>>#Running Sample.py
>>>from Sample import *
>>>
Press the Var key and select 'coin_flip()'
In the ( ), enter a number of flips!
>>>coin_flip(10)
10 coin flips: Heads: 4 Tails: 6
>>>
```

## Plotting

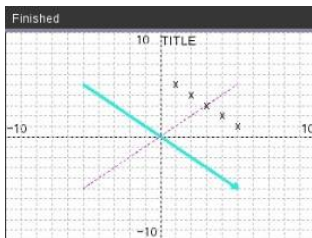
```
# Plotting example
import ti_plotlib as plt

# Set up the graph window
plt.window(-10,10,-10,10)
plt.axes("on")
plt.grid(1,1,"dashed")
# Add leading spaces to position the title
plt.title("          TITLE")

# Set the pen style and the graph color
plt.pen("medium","solid")
plt.color(28,242,221)
plt.line(-5,5,5,-5,"arrow")

plt.pen("thin","dashed")
plt.color(224,54,243)
plt.line(-5,-5,5,5,"")

# Scatter plot from 2 lists
plt.color(0,0,0)
xlist=[1,2,3,4,5]
ylist=[5,4,3,2,1]
plt.scatter(xlist,ylist, "x")
```



## Drawing

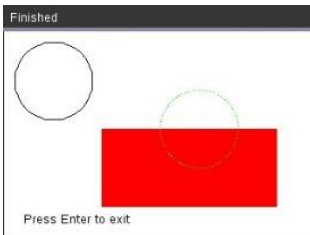
```
from ti_draw import *

# (0,0) is in top left corner of screen
# Let's draw some circles and squares
# Circle with center at (50,50) and radius 40
draw_circle(50,50,40)

# Set color to red (255,0,0) and fill a rectangle of
# of width 180, height 80 with top left corner at
# (100,100)
set_color(255,0,0)
fill_rect(100,100,180,80)

# Set color to green and pen style to "thin"
# and "dotted".
# Then, draw a circle with center at (200,100)
# and radius 40
set_color(0,255,0)
set_pen("thin","dotted")
draw_circle(200,100,40)

set_color(0,0,0)
draw_text(20,200,"Press Enter to exit")
```



## Image

```
# Image Processing
#=====
from ti_image import *
from ti_draw import *
#=====

# Load and show the 'manhole_cover' image
# It's in a Notes app
# Draw a circle on top
iml=load_image("manhole_cover")
iml.show_image(0,0)
set_color(0,255,0)
set_pen("thick","dashed")
draw_circle(140,110,100)
```



## Hub

このプログラムは、Pythonを使って、プログラム可能なマイクロコントローラーであるTI-Innovator™ Hubを制御します。TI-Innovator™ Hubを接続せずにプログラムを実行すると、エラーメッセージが表示されます。

TI-Innovator™ Hubの詳細については、[education.ti.com](http://education.ti.com)にアクセスしてください。

```
##### Import Section #####
from ti_hub import *
from math import *
from random import *
from time import sleep
from ti_plotlib import text_at,cls
from ti_system import get_key
##### End of Import Section #####

print("Connect the TI-Innovator Hub and hit 'enter'")
input()
print("Blinking the RGB LED for 4 seconds")
# Set the RGB LED on the Hub to purple
color.rgb(255,0,255)

# Blink the LED 2 times a second for 4 seconds
color.blink(2,4)

sleep(5)
print("The brightness sensor reading is: ", brightness.measurement())

# Generate 10 random colors for the RGB LED
# Play a tone on the Hub based on the random
# color
print("Generate 10 random colors on the Hub & play a tone")
for i in range(10):
    •r=randint(0,255)
    •b=randint(0,255)
    •g=randint(0,255)
    •color.rgb(r,g,b)
    •sound.tone((r+g+b)/3,1)
    •sleep(1)

color.off()
```

## General Information

### *Online Help*

[education.ti.com/eguide](http://education.ti.com/eguide)

Select your country for more product information. **Contact TI Support**

[education.ti.com/ti-cares](http://education.ti.com/ti-cares)

Select your country for technical and other support resources.

### *Service and Warranty Information*

[education.ti.com/warranty](http://education.ti.com/warranty)

Select your country for information about the length and terms of the warranty or about product service.

Limited Warranty. This warranty does not affect your statutory rights.

Texas Instruments Incorporated 12500 TI Blvd.

Dallas, TX 75243



TI-Nspire™ Python プログラミングガイドブック

---

2021年12月24日 第1刷 発行

編集 株式会社 ナオコ  
発行者 中澤房紀  
発行所 株式会社 ナオコ  
〒160-0023 東京都新宿区西新宿3-9-2  
イマス西新宿第一ビル5階  
Tel:03-5309-2880 Fax:03-5309-2881  
ホームページ [www.naoco.com](http://www.naoco.com)  
メール [ti-calc@naoco.com](mailto:ti-calc@naoco.com)

---

落丁・乱丁本はお取り替えいたします。

Printed in Japan