Texas Instruments

TI-Nspire™ Python Programming Guidebook

Learn more about TI Technology through the online help at education.ti.com/eguide.

Important Information

Contents

Getting Started with Python Programming

Using Python with TI-Nspire™ products you can:

- add Python programs to TNS files
- create Python programs using templates
- interact and share data with other TI-Nspire™ apps
- interact with the TI-Innovator™ Hub and TI-Innovator™ Rover

The TI-Nspire™ Python implementation is based on MicroPython, which is a small subset of the Python 3 standard library designed to run on microcontrollers. The original MicroPython implementation has been adapted for use by TI.

Note: Some numeric answers may vary from the Calculator results due to differences in the underlying maths implementations.

Python is available on these TI-Nspire™ products:

Handhelds	Desktop Software
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: In most cases, functionality is identical between the handheld and the software views, but you may see some differences. This guide assumes that you are using the handheld device or the Handheld view in the software.

Python Modules

TI-Nspire™ Python includes the following modules:

Standard Modules	TI Modules
Math (maths)	TI PlotLib (ti_plotlib)
Random (random)	TI Hub (ti_hub)
Complex Math (cmath)	TI Rover (ti_rover)
Time (time)	TI System (ti_system)
	TI Draw (ti_draw)
	TI Image (ti_image)

Note: If you have existing Python programs created in other Python development environments, you may need to edit them to run on the TI-Nspire™ Python solution. Modules may use different methods, arguments, and ordering of methods in a program compared to the TI modules. In general, be aware of compatibility when using any version of Python and Python modules.

When transferring Python programs from a non-TI platform to a TI platform OR from one TI product to another, remember:

- Programs that use core language features and standard libs (math, random, etc.) can be ported without changes.
- Programs that use platform-specific libraries such as matplotlib for PC or TI modules will require edits before they will run on a different platform. This may be true even between TI platforms.

As with any version of Python, you will need to include imports to use any functions, methods, or constants contained in a given module. For example, to execute the cos() function from the maths module, use the following commands:

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

For a list of menus with their items and descriptions, please see the Menu Map section.

Installing a Python programme as a module

To save your Python programme as a module:

- In the Editor, select **Actions > Install as Python Module**.
- In the Shell, select **Tools > Install as Python Module**.

After selection, the following occurs:

- The Python syntax is checked.
- The file is saved and moved to the PyLib folder.
- A dialogue appears confirming that the file has been installed as a module.
- The file is closed and the module is ready for use.
- The module name will be added to the **More Modules** menu with a **from <module> import*** menu item.

If you plan to share this module with others, it is recommended that you follow these guidelines:

- Store only one module per TNS file.
- The module name matches the name of TNS file (e.g. "my_program" module is in the "my_program.tns" file).
- Add a Notes page before the Python Editor that describes the intent of the module, the version, and the functions.
- Use the ver() function to display the version number of the module.
- (Optional) Add a help function to display the list of methods in the function.

Python Workspaces

There are two workspaces for your Python programming: The Python Editor and the Python Shell.

Python Editor	Python Shell
• Create, edit and save Python programs • Syntax highlighting and auto-indentation • Inline prompts to guide with function arguments • Tooltips to show range of valid values • var key lists global user variables and functions defined in the current program • Keypad shortcuts	• Run Python programs • Convenient for testing small code fragments • Interaction with Shell history to select previous inputs and outputs for re-use • var key lists global user variables defined in the last program ran in the given problem

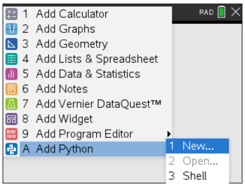
Note: Multiple Python programs and Shells can be added to a problem.

Python Editor

The Python Editor is where you can create, edit and save Python programs.

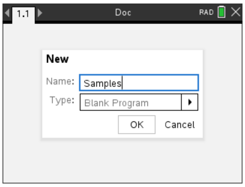
Adding a Python Editor page

To add a new Python Editor page in the current problem, press menu and select **Add Python > New**.



You can create a blank program, or you can select a template.

Blank Program



1.1 Doc

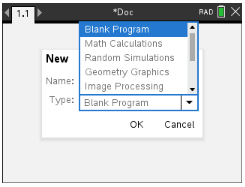
New

Name: Samples

Type: Blank Program

OK Cancel

Template



1.1 *Doc

New

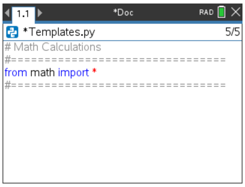
Name:

Type: Blank Program

After creating the program, the Python Editor is displayed. If you selected a template, the necessary import statements are automatically added (see below).

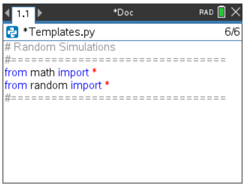
Note: You can have multiple programs in a single TNS file just like other apps. If the Python program is intended to be used as a module, the TNS file can be saved in the PyLib folder. That module can then be used in other programs and documents.

Maths Calculations



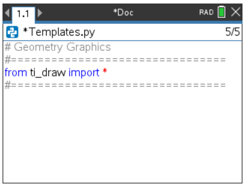
```
*Templates.py                              5/5
# Math Calculations
#================================
from math import *
#================================
```

Random Simulations



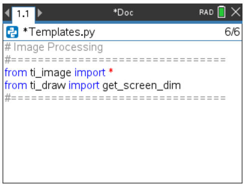
```
*Templates.py                              6/6
# Random Simulations
#================================
from math import *
from random import *
#================================
```

Geometry Graphics



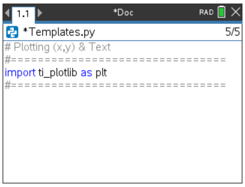
```
*Templates.py                              5/5
# Geometry Graphics
#================================
from ti_draw import *
#================================
```

Image Processing



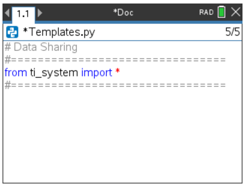
```
*Templates.py                              6/6
# Image Processing
#================================
from ti_image import *
from ti_draw import get_screen_dim
#================================
```

Plotting (x,y) & Text



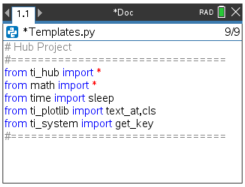
```
*Templates.py                              5/5
# Plotting (x,y) & Text
#================================
import ti_plotlib as plt
#================================
```

Data Sharing



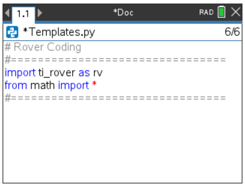
```
*Templates.py                              5/5
# Data Sharing
#================================
from ti_system import *
#================================
```

TI-Innovator Hub Project



```
*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
#================================
```

TI-Rover Coding



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

Opening a Python Program

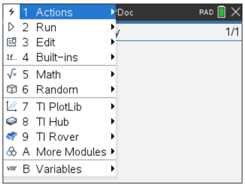
To open an existing Python program, press doc▾ and select **Insert > Add Python > Open**. This will display a list of programs that have been saved in the TNS file.

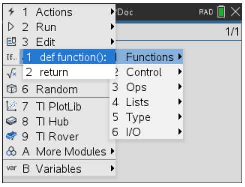
If the Editor page used to create the program has been deleted, the program is still available in the TNS file.

Working in the Python Editor

Pressing menu displays the Document Tools menu. With these menu options you can add, move, and copy blocks of code for your program.

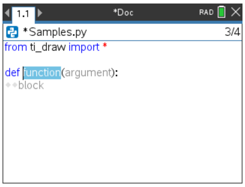
Document Tools menu





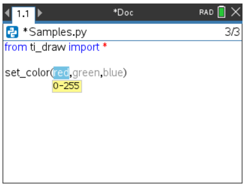
Items selected from the module menus will automatically add a code template to the Editor with inline prompts for each part of the function. You can navigate from one argument to the next by pressing tab (forward) or ⇧shift+tab (backward). Tooltips or pop-up lists will appear when available to help you select the proper values.

Inline prompts



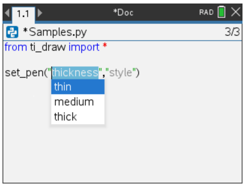
```
from ti_draw import *

def function(argument):
    block
```

Tooltips



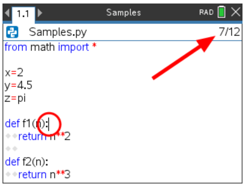
```
from ti_draw import *

set_color(red,green,blue)
```

Pop-up lists



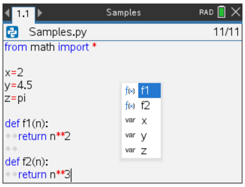
```
from ti_draw import *

set_pen("thickness","style")
```

The numbers to the right of the program name reflect the current line number of the cursor and the total number of lines in the program.



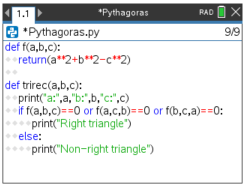
```
1.1   Samples   RAD
Samples.py   7/12
from math import *

x=2
y=4.5
z=pi

def f1(n):
  return n**2

def f2(n):
  return n**3
```

Global functions and variables defined in the lines above the current cursor position can be inserted by pressing var and selecting from the list.



```
1.1   Samples   RAD
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
```

As you add code to your program, the Editor displays keywords, operators, comments, strings and indents in different colours to aid in identifying the different elements.

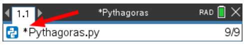


```
1.1   *Pythagoras   RAD
*Pythagoras.py   9/9
def f(a,b,c):
  return(a**2+b**2-c**2)

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")
```

Saving and running programs

When your are finished with your program, press menu and select **Run > Check Syntax & Save**. This will check the syntax of the Python program and save it to the TNS file.

Note: If you have unsaved changes in your program, an asterisk will display next to the program name.



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

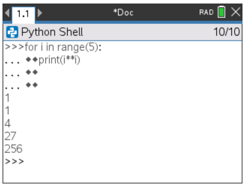
To run the program, press menu and select **Run > Run**. This will run the current program in the next Python Shell page or a new one if the next page is not a Shell.

Note: Running the program automatically checks the syntax and saves the program.

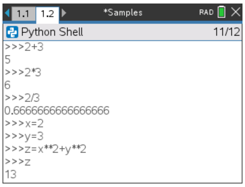
Python Shell

The Python Shell is the interpreter that executes your Python programs, other pieces of Python code, or simple commands.

Python code *Simple commands*



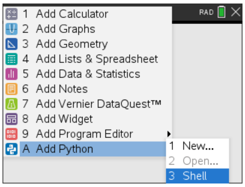
```
>>>for i in range(5):
... ••print(i**i)
... ••
... ••
1
1
4
27
256
>>>
```



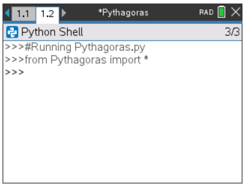
```
>>>2+3
5
>>>2*3
6
>>>2/3
0.6666666666666666
>>>x=2
>>>y=3
>>>z=x**2+y**2
>>>z
13
```

Adding a Python Shell page

To add a new Python Shell page in the current problem, press [menu] and select **Add Python > Shell**.



The Python Shell can also be launched from the Python Editor by executing a program by pressing [menu] and selecting **Run > Run**.

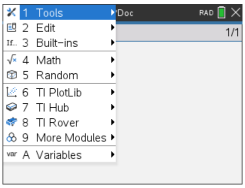


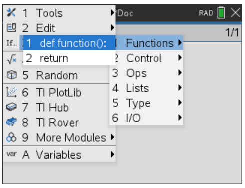
```
>>>#Running Pythagoras.py
>>>from Pythagoras import *
>>>
```

Working in the Python Shell

Pressing [menu] displays the Document Tools menu. With these menu options you can add, move, and copy blocks of code.

Document Tools menu

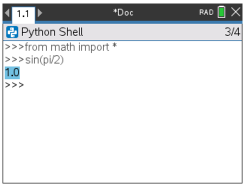




Note: If you use any method from one of the available modules, be sure to execute an import module statement first as in any Python coding environment.

Interaction with the Shell output is similar to the Calculator app where you can select and copy previous inputs and outputs for use elsewhere in the Shell, Editor, or other apps.

Up arrow to select, then copy and paste to the desired location

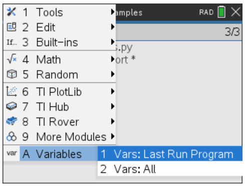


```
>>>from math import *
>>>sin(pi/2)
1.0
>>>
```

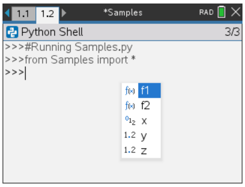
Global functions and variables from the last run program can be inserted by pressing var or ctrl+L and selecting from the list or press menu and select **Variables > Vars: Last Run Program**.

To choose from a list of global functions and variables from both the last run program and any imported modules, press menu and select **Variables > Vars: All**.

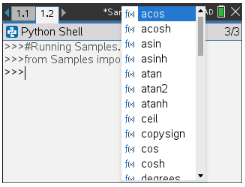
Variables menu



Last Run Program variables *All variables*



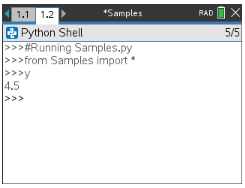
```
>>>#Running Samples.py
>>>from Samples import *
>>>|
```



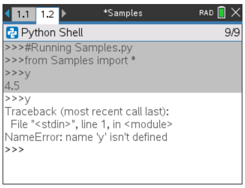
```
>>>#Running Samples.
>>>from Samples impo
>>>|
```

All Python Shell pages in the same problem share the same state (user-defined and imported variable definitions). When you save or run a Python program in that problem, or press menu and select **Tools > Reinitialize Shell**, the Shell history will then have a grey background indicating that the previous state is no longer valid.

Before saving or reinitialising *After saving or reinitialising*



```
>>>#Running Samples.py
>>>from Samples import *
>>>y
4.5
>>>
```



```
>>>#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: The menu **Tools > Clear History** option clears the screen of any previous activity in the Shell, but variables are still available.

Messages

Error and other informational messages may display while you are in a Python session. If an error is displayed in the Shell when a program executes, a program line number will display. Press ctrl menu and select **Go to Python Editor**. In the Editor, press menu then select **Edit > Go to Line**. Enter the line number and press enter. The cursor will display on the first character of the line where the error occurred.

Interrupting a Running Program

While a programme or function is running, the busy pointer ⌚ is displayed.

- To stop the programme or function,
  - Windows®: Press the **F12** key.
  - Mac®: Press the **F5** key.
  - Handheld: Press the ⌂on key.

Python Menu Map

This section lists all of the menus and menu items for the Python Editor and Shell and a brief description for each one.

Note: For the menu items that have keyboard shortcuts, Mac® users should substitute ⌘ (**cmd**) anywhere **Ctrl** is used. For a complete list of TI-Nspire™ handheld and software shortcuts, see the TI-Nspire™ Technology eGuide.

Actions Menu

Note: This applies to the Editor only.

Item	Description
New	Opens the **New** dialog box where you enter a name and select a type for your new program.
Open	Opens a list of programs available in the current document.
Create Copy	Opens the **Create Copy** dialogue box where you can save the current program under another name.
Rename	Opens the **Rename** dialogue box where you can rename the current program.
Close	Closes the current program.
Settings	Opens the **Settings** dialogue box where you can change the font size for both the Editor and Shell.
Install as Python module	Cheques the Python syntax of the current TNS file and moves it to the PyLib folder.

Run Menu

Note: This applies to the Editor only.

Item	**Shortcut**	**Description**
Run	Ctrl+R	Checks syntax, saves program, and executes in a Python Shell.
Check Syntax & Save	Ctrl+B	Checks syntax and saves program.
Go to Shell	N/A	Shifts focus to the Shell related to the current program or opens a new Shell page next to the Editor.

Tools Menu

Note: This applies to the Shell only.

Item	Shortcut	Description
Rerun Last Program	Ctrl+R	Reruns the last program related to the current Shell.
Go to Python Editor	N/A	Opens the Editor page related to the current Shell.
Run	N/A	Opens a list of programs available in the current document. After selection, the chosen program is run.
Clear History	N/A	Clears the history in the current Shell but does not re-initialise the Shell.
Re-initialise Shell	N/A	Resets the state of all open Shell pages in the current problem. All defined variables and imported functions are no longer available.
dir()	N/A	Displays list of functions in the specified module when used after the import statement.
From PROGRAM import *	N/A	Opens a list of programs available in the current document. After selection, the import statement is pasted in the Shell.
Install as Python Module	N/A	Enabled only for modules in binary format. Moves the current TNS file to the PyLib folder.

Edit Menu

Note: Ctrl+A selects all lines of code or output for cutting or deleting (Editor only), or copying and pasting (Editor and Shell).

Item	Shortcut	Description
Indent	TAB*	Indents text on the current line or selected lines. * If there are incomplete inline prompts, TAB will navigate to the next prompt.
Dedent	Shift+TAB**	Dedents text on the current line or selected lines. ** If there are incomplete inline prompts, Shift+TAB will navigate to the previous prompt.
Comment/Uncomment	Ctrl+T	Adds/removes comment symbol to/from the beginning of the current line.
Insert Multi-line String	N/A	(Editor only) Inserts multi-line string template.
Find	Ctrl+F	(Editor only) Opens **Find** dialogue box and searches for the entered string in the current program.
Replace	Ctrl+H	(Editor only) Opens **Replace** dialogue box and searches for the entered string in the current program.
Go to Line	Ctrl+G	(Editor only) Opens **Go to Line** dialogue box and jumps to the specified line in the current program.
Beginning of Line	Ctrl+8	Moves cursor to the beginning of the current line.
End of Line	Ctrl+2	Moves cursor to the end of the current line.
Jump to Top	Ctrl+7	Moves cursor to the beginning of the first line in the program.
Jump to Bottom	Ctrl+1	Moves cursor to the end of the last line in the program.

Built-ins Menu

Functions

Item	Description
def function():	Defines a function dependent on specified variables.
return	Defines the value produced by a function.

Control

Item	Description
if..	Conditional statement.
if..else..	Conditional statement.
if..elif..else..	Conditional statement.
for index in range(size):	Iterates over a range.
for index in range(start,stop):	Iterates over a range.
for index in range(start,stop,step):	Iterates over a range.
for index in list:	Iterates over list elements.
while..	Executes statements in a code block until a condition evaluates to False.
elif:	Conditional statement.
else:	Conditional statement.

Ops

Item	Description
x=y	Sets variable value.
x==y	Pastes equal to (==) comparison operator.
x!=y	Pastes not equal to (!=) comparison operator.
x>y	Pastes greater than (>) comparison operator.
x>=y	Pastes greater than or equal to (>=) comparison operator.
x<y	Pastes less than (<) comparison operator.
x<=y	Pastes less than or equal to (<=) comparison operator.

Item	Description
and	Pastes and (and) logical operator.
or	Pastes or (or) logical operator.
not	Pastes not (not) logical operator.
True	Pastes True Boolean value.
False	Pastes False Boolean value.

Lists

Item	Description
[]	Pastes brackets ([]).
list()	Converts sequence into "list" type.
len()	Returns number of elements of the list.
max()	Returns maximum value in the list.
min()	Returns minimum value in the list.
.append()	The method appends an element to a list.
.remove()	The method removes the first instance of an element from a list.
range(start,stop,step)	Returns a set of numbers.
for index in range(start,stop,step)	Used to iterate over a range.
.insert()	The method adds an element at the specified position.
.split()	The method returns a list with elements separated by specified delimiter.
sum()	Returns sum of the elements of a list.
sorted()	Returns a sorted list.
.sort()	The method sorts a list in place.

Type

Item	Description
int()	Returns an integer part.
float()	Returns a float value.

Item	Description
round(x,ndigits)	Returns a floating point number rounded to the specified number of digits.
str()	Returns a string.
complex()	Returns a complex number.
type()	Returns the type of the object.

I/O

Item	Description
print()	Displays argument as string.
input()	Prompts user for input.
eval()	Evaluates an expression represented as a string.
.format()	The method formats the specified string.

Math Menu

Note: When creating a new program that uses this module, it is recommended to use the **Math Calculations** program type. This will ensure that all the relevant modules are imported.

Item	Description
from math import *	Imports all methods (functions) from the maths module.
fabs()	Returns absolute value of a real number.
sqrt()	Returns square root of a real number.
exp()	Returns e**x.
pow(x,y)	Returns x raised to the power y.
log(x,base)	Returns $\log_{base}$(x). log(x) with no base returns the natural logarithm x.
fmod(x,y)	Returns module value of x and y. Use when x and y are floats.
ceil()	Returns the smallest integer greater than or equal to a real number.
floor()	Returns the largest integer less than or equal to a real number.
trunc()	Truncates a real number to an integer.
frexp()	Returns a pair (y,n) where x == y * 2**n.

Const

Item	Description
e	Returns value for the constant e.
pi	Returns value for the constant pi.

Trig

Item	Description
radians()	Converts angle in degrees to radians.
degrees()	Converts angle in radians to degrees.
sin()	Returns sine of argument in radians.

Item	Description
cos()	Returns cosine of argument in radians.
tan()	Returns tangent of argument in radians.
asin()	Returns arc sine of argument in radians.
acos()	Returns arc cosine of argument in radians.
atan()	Returns arc tangent of argument in radians.
atan2(y,x)	Returns arc tangent of y/x in radians.

Random Menu

Note: When creating a new program that uses this module, it is recommended to use the **Random Simulations** program type. This will ensure that all the relevant modules are imported.

Item	Description
from random import*	Imports all methods from the random module.
random()	Returns a floating point number from 0 to 1.0.
uniform(min,max)	Returns a random number x (float) such that min <= x <= max.
randint(min,max)	Returns a random integer between min and max.
choice(sequence)	Returns a random element from a non-empty sequence.
randrange(start,stop,step)	Returns a random number from start to stop by step.
seed()	Initialises random number generator.

TI PlotLib Menu

Note: When creating a new program that uses this module, it is recommended to use the **Plotting (x,y) & Text** program type. This will ensure that all the relevant modules are imported.

Item	Description
import ti_plotlib as plt	Imports all methods (functions) from the ti_plotlib module in the "plt" namespace. As a result, all function names pasted from the menus will be preceded by "plt.".

Setup

Item	Description
cls()	Clears the plotting canvas.
grid(x-scale,y-scale,"style")	Displays a grid using specified scale for x and y axes.
window(xmin,xmax,ymin,ymax)	Defines the plotting window by mapping the the specified horizontal interval (xmin, xmax) and vertical interval (ymin, ymax) to the allotted plotting area (pixels).
auto_window(x-list,y-list)	Autoscales the plotting window to fit the data ranges within x-list and y-list specified in the program prior to the auto_window().
axes("mode")	Displays axes on specified window in the plotting area.
labels("x-label","y-label",x,y)	Displays "x-label" and "y-label" labels on the plot axes at row positions x and y.
title("title")	Displays "title" centred on top line of window.
show_plot()	Displays the buffered drawing output. The use_buffer() and show_plot() functions are useful in cases where displaying multiple objects on the screen could cause delays (not necessary in most cases).
use_buffer()	Enables an off-screen buffer to speed up drawing.

Draw

Item	Description
colour(red,green,blue)	Sets the colour for all following graphics/plotting.
cls()	Clears the plotting canvas.
show_plot()	Executes the display of the plot as set up in the program.
scatter(x-list,y-list,"mark")	Plots a sequence of ordered pair from (x-list,y-list) with the specified mark style.
plot(x-list,y-list,"mark")	Plots a line using ordered pairs from specified x-list and y-list.
plot(x,y,"mark")	Plots a point using coordinates x and y with the specified mark style.
line(x1,y1,x2,y2,"mode")	Plots a line segment from (x1,y1) to (x2,y2).
lin_reg(x-list,y-list,"display")	Calculates and draws the linear regression model, axe+b, of x-list,y-list.
pen("size","style")	Sets the appearance of all following lines until the next pen() is executed.
text_at(row,"text","align")	Displays "text" in plotting area at specified "align".

Properties

Item	Description
xmin	Specified variable for window arguments defined as plt.xmin.
xmax	Specified variable for window arguments defined as plt.xmax.
ymin	Specified variable for window arguments defined as plt.ymin.
ymax	Specified variable for window arguments defined as plt.ymax.
m	After plt.linreg() is executed in a program, the computed values of slope, m, and intercept, b, are stored in plt.m and plt.b.
b	After plt.linreg() is executed in a program, the computed values of slope, a, and intercept, b, are stored in plt.a and plt.b.

TI Hub Menu

Note: When creating a new program that uses this module, it is recommended to use the **Hub Project** program type. This will ensure that all the relevant modules are imported.

Item	Description
from ti_hub import*	Imports all methods from the ti_hub module.

Hub Built-in Devices > Colour Output

Item	Description
rgb(red,green,blue)	Sets the colour for the RGB LED.
blink(frequency,time)	Sets the blinking frequency and duration for the selected colour.
off()	Turns the RGB LED off.

Hub Built-in Devices > Light Output

Item	Description
on()	Turns the LED on.
off()	Turns the LED off.
blink(frequency,time)	Sets the blinking frequency and duration for the LED.

Hub Built-in Devices > Sound Output

Item	Description
tone(frequency,time)	Plays a tone of the specified frequency for the specified time.
note("note",time)	Plays the specified note for the specified time. The note is specified using the note name and an octave. For example: A4, C5. The note names are C, CS, D, DS, E, F, FS, G, GS, A, AS, and B. The octave numbers range from 1 to 9 (inclusive).

Item	Description
tone(frequency,time,tempo)	Plays a tone of the specified frequency for the specified time and tempo. The tempo defines the number of beeps per second ranging from 0 to 10 (inclusive).
note("note",time,tempo)	Plays the specified note for the specified time and tempo. The note is specified using the note name and an octave. For example: A4, C5. The note names are C, CS, D, DS, E, F, FS, G, GS, A, AS, and B. The octave numbers range from 1 to 9 (inclusive). The tempo numbers range from 0 to 10 (inclusive).

Hub Built-in Devices > Brightness Input

Item	Description
measurement()	Reads the built-in BRIGHTNESS (light level) sensor and returns a reading. The default range is 0 to 100. This can be changed using the range() function.
range(min,max)	Sets the range for mapping the readings from the light level sensor. If both are missing, or set to a value of None, then the default brightness range of 0 to 100 is set.

Add Input Device

This menu has a list of the sensors (input devices) supported by the ti_hub module. All the menu items will paste the name of the object and expect a variable and a port used with the sensor. Each sensor has a measurement() method that returns the value of the sensor.

Item	Description
DHT (Digital Humidity & Temp)	Returns a list consisting of the current temperature, humidity, type of sensor, and last cached read status.
Ranger	Returns the current distance measurement from the specified ultrasonic ranger. • **measurement_time()** - Returns the time that the ultrasonic signal takes to reach the object (the "time of flight").

Item	Description
Light Level	Returns the brightness level from the external light level (brightness) sensor.
Temperature	Returns the temperature reading from the external temperature sensor. The default configuration is to support the Seeed temperature sensor in IN 1, IN 2 or IN 3 ports. To use the TI LM19 Temperature sensor from the TI-Innovator™ Hub breadboard pack, edit the port to the BB pin in use and use an optional argument "TIANALOG". Example: mylm19=temperature("BB 5","TIANALOG")
Moisture	Returns the moisture sensor reading.
Magnetic	Detects the presence of a magnetic field. The threshold value to determine the presence of the field is set through the trigger() function. The default value of the threshold is 150.
Vernier	Reads the value from the Vernier analogue sensor specified in the command. The command supports the following Vernier sensors: • **temperature** - Stainless Steel Temperature sensor. • **lightlevel** - TI Light level sensor. • **pressure** - Original petrol pressure sensor • **pressure** - Newer petrol pressure sensor. • **pH** - pH sensor. • **force10** - ±10 N setting, Dual Force Sensor. • **force50** - ±50 N setting, Dual Force Sensor. • **accelerometer** - Low-G Accelerometer. • **generic** - Allows setting of other sensors not supported directly above, and use of the calibrate() API above to set equation coefficients.
Analogue In	Supports the use of analogue input generic devices.
Digital In	Returns the current state of the digital pin connected to the DIGITAL object, or the cached state of the digital output value last SET to the object.
Potentiometer	Supports a potentiometer sensor. The range of the sensor can be changed by the range

Item	Description
	() function.
Thermistor	Reads thermistor sensors. The default coefficients are designed to match the thermistor included in the Breadboard Pack of the TI-Innovator™ Hub, when used with a 10KΩ fixed resistor. A new set of calibration coefficients and reference resistance for the thermistor can be configured using the calibrate() function.
Loudness	Supports sound loudness sensors.
Colour Input	Provides interfaces to an I2C-connected Colour Input sensor. The bb_port pin is used in addition to the I2C port to control the LED on the colour sensor. • **colour_number():** Returns a value from 1 to 9 that represents the colour the sensor is detecting. The numbers represent the colours per the following mapping: 1: Red 2: Green 3: Blue 4: Cyan 5: Magenta 6: Yellow 7: Black 8: White 9: Gray • **red():** Returns a value from 0 to 255 that represents the intensity of the RED colour level being detected. • **green():** Returns a value from 0 to 255 that represents the intensity of the GREEN colour level being detected. • **blue():** Returns a value from 0 to 255 that represents the intensity of the BLUE colour level being detected. • **gray():** Returns a value from 0 to 255 that represents the grey level being detected, where 0 is black and 255 is white.

Item	Description
BB Port	Provides support for using all 10 BB port pins as a combined digital input/output port. The initialisation functions have an optional "mask" parameter that allows the use of the subset of the 10 pins. • **read_port():** Reads the current values on the input pins of the BB port. • **write_port(value):** Sets the output pin values to the specified value, where value is between 0 and 1023. Note that the value is also adjusted against the mask value in the var=bbport(mask) operation, if a mask was provided.
Hub Time	Provides access to the internal millisecond timer.
TI-RGB Array	Provides functions for programming the TI-RGB Array. The initialisation function accepts an optional "LAMP" parameter to enable a high-brightness mode for the TI-RGB Array that requires an external power supply. • **set(led_position, r,g,b):** Sets a specific led_ position (0-15) to the specified r,g,b value, where r,g,b are values from 0 to 255. • **set(led_list,red,green,blue):** Sets the LEDs defined in the "led_list" to the colour specified by "red", "green", "blue". The "led_list" is a Python list that includes indexes of the LEDs from 0 to 15. For example, the set([0,2,4,6,15], 0, 0, 255) will set LEDs 0, 2, 4, 6 and 15 to blue. • **set_all(r,g,b):** Sets all RGB LEDs in the array to the same r,g,b value. • **all_off():** Turns off all RGBs in the array. • **measurement():** Returns the approximate current draw that the RGB array is using from the TI-Innovator™ in milliAmps. • **pattern(pattern):** Using the value of the argument as a binary value in the range 0 to 65535, turns on pixels where a 1 value in the representation would be. LEDs are turned on as RED with pwm level value of 255. • **pattern(value,red,green,blue):** Sets the LEDs defined by the "pattern" to the colour specified by "red", "green", "blue".

Add Output Device

This menu has a list of the output devices supported by the ti_hub module. All the menu items will paste the name of the object and expect a variable and a port used with the device.

Item	Description
LED	Functions for controlling externally connected LEDs.
RGB	Support for controlling external RGB LEDs.
TI-RGB Array	Provides functions for programming the TI-RGB Array.
Speaker	Functions for supporting an external speaker with the TI-Innovator™ Hub. The functions are the same as the ones for "sound" above.
Power	Functions for controlling external power with the TI-Innovator™ Hub. • **set(value):** Sets the Power level to the specified value, between 0 and 100. • **on():** Sets the Power level to 100. • **off():** Sets the Power level to 0.
Continuous Servo	Functions for controlling continuous servo engines. • **set_cw(speed,time):** The servo will spin in the clockwise direction at the specified speed (0-255) and for the specific duration in seconds. • **set_ccw(speed,time):** The servo will spin in the counter-clockwise direction at the specified speed (0-255) and for the specific duration in seconds. • **stop():** Stops the continuous servo.
Analogue Out	Functions for the use of analogue input generic devices.
Vibration Engine	Functions for controlling vibration engines. • **set(val):** Sets the vibration engine intensity to "val" (0-255). • **off():** Turns the vibration engine off. • **on():** Turns the vibration engine on at the highest level.
Relay	Controls interfaces for controlling relays. • **on():** Sets the relay to the ON state. • **off():** Sets the relay to the OFF state.
Servo	Functions for controlling servo engines. • **set_position(pos):** Sets the sweep servo position within a range of -90 to +90. • **zero():** Sets the sweep servo to the zero position.
Squarewave	Functions for generating a square wave.

Item	Description
	• **set(frequency,duty,time):** Sets the output squarewave with a default duty cycle of 50% (if duty is not specified) and an output frequency specified by "frequence". The frequency may be from 1 to 500 Hz. The duty cycle, if specified, may be from 0 to 100%. • **off():** Turns the squarewave off.
Digital Out	Interfaces for controlling a digital output. • **set(val):** Sets the digital output to the value specified by "val" (0 or 1). • **on():** Sets the state of the digital output to high (1). • **off():** Sets the state of the digital output to low (0).
BB Port	Provides functions for programming the TI-RGB Array. See the details above.

Commands

Item	Description
sleep(seconds)	Pauses the program for the specified number of seconds. Imported from the 'time' module.
text_at(row,"text","align")	Displays the specified "text" in the plotting area at specified "align". Part of the ti_plotlib module.
cls()	Clears the Shell screen for plotting. Part of the ti_plotlib module.
while get_key() != "esc":	Runs the commands in the "while" loop until the "esc" key is pressed.
get_key()	Returns a string representing the key pressed. The '1' key returns "1", 'esc' returns "esc", and so on. When called without any parameters - get_key() - it returns immediately. When called with a parameter - get_key(1) - it waits until a key is pressed. Part of the ti_system module.

Ports

These are the input and output ports available on the TI-Innovator™ Hub.

Item
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 Menu

Note: When creating a new program that uses this module, it is recommended to use the **Rover Coding** program type. This will ensure that all the relevant modules are imported.

Item	Description
import ti_rover as rv	Imports all methods (functions) from the ti_rover module in the "rv" namespace. As a result, all function names pasted from the menus will be preceded by "rv.".

Drive

Item	Description
forward(distance)	Moves Rover forward the specified distance in grid units.
backward(distance)	Moves Rover backward the specified distance in grid units.
left(angle_degrees)	Turns Rover left the specified angle in degrees.
right(angle_degrees)	Turns Rover right the specified angle in degrees.
stop()	Stops any current movement immediately.
stop_clear()	Stops any current movement immediately and clears all pending commands.
resume()	Resumes the processing of commands.
stay(time)	Rover stays in place for the specified amount of time in seconds (optional). If no time is specified, the Rover stays for 30 seconds.
to_xy(x,y)	Moves Rover to coordinate position (x,y) on virtual grid.
to_polar(r,theta_degrees)	Moves Rover to polar coordinate position (r, theta) on virtual grid. The angle is specified in degrees.
to_angle(angle,"unit")	Spins Rover to the specified angle in the virtual grid. The angle is relative to a zero angle which points down the x-axis in the virtual grid.

Drive > Drive with Options

Item	Description
forward_time(time)	Moves Rover forward for the specified time.
backward_time(time)	Moves Rover backward for the specified time.
forward(distance,"unit")	Moves Rover forward at the default speed for the specified distance. The distance can be specified in grid units, metres, or wheel revolutions.
backward(distance,"unit")	Moves Rover backward at the default speed for the specified distance. The distance can be specified in grid units, metres, or wheel revolutions.
left(angle,"unit")	Turns Rover left the specified angle. The angle can be in degrees, radians, or gradians.
right(angle,"unit")	Turns Rover right the specified angle. The angle can be in degrees, radians, or gradians.
forward_time(time,speed,"rate")	Moves Rover forward for the specified time at the specified speed. The speed can be specified in grid units/s, metres/s, or wheel revolutions/s.
backward_time(time,speed,"rate")	Moves Rover backward for the specified time at the specified speed. The speed can be specified in grid units/s, metres/s, or wheel revolutions/s.
forward(distance,"unit",speed,"rate")	Moves Rover forward for the specified distance at the specified speed. The distance can be specified in grid units, metres, or wheel revolutions. The speed can be specified in grid units/s, metres/s, or wheel revolutions/s.
backward(distance,"unit",speed,"rate")	Moves Rover backward for the specified distance at the specified speed. The distance can be specified in grid units, metres, or wheel revolutions. The speed can be specified in grid units/s, metres/s, or wheel revolutions/s.

Inputs

Item	Description
ranger_measurement()	Reads the ultrasonic distance sensor on the front of the Rover, returning the current distance in metres.
colour_measurement()	Returns a value from 1 to 9, indicating the predominant colour being "seen" by the Rover colour input sensor. 1 = red 2 = green 3 = blue 4 = cyan 5 = magenta 6 = yellow 7 = black 8 = grey 9 = white
red_measurement()	Returns a value between 0 and 255 that indicates the perceived red level being seen by the colour input sensor.
green_measurement()	Returns a value between 0 and 255 that indicates the perceived green level being seen by the colour input sensor.
blue_measurement()	Returns a value between 0 and 255 that indicates the perceived blue level being seen by the colour input sensor.
grey_measurement()	Returns a value between 0 and 255 that indicates the perceived grey level being seen by the colour input sensor.
encoders_gyro_measurement()	Returns a list of values that contains the left and right wheel encoder counts as well as the current gyro heading.
gyro_measurement()	Returns a value that represents the current gyro reading, including drift, in the degrees.
ranger_time()	Returns the time that the ultrasonic signal from the TI-Rover ranger takes to reach the object (the "time of flight").

Outputs

Item	Description
color_rgb(r,g,b)	Sets the colour of the Rover RGB LED to the specific red, green, blue values.
color_blink(frequency,time)	Sets the blinking frequency and duration for the selected colour.
color_off()	Turns the Rover RGB LED off.
engine_left(speed,time)	Sets the left engine power to the specified value for the specified duration. The speed is in the range -255 to 255 with 0 being stop. Positive speed values are counter-clockwise rotation, and negative speed values are clockwise. The optional time parameter, if specified, has a valid range of 0.05 to 655.35 seconds. If not specified, a default of 5 seconds is used.
engine_right(speed,time)	Sets the left engine power to the specified value for the specified duration. The speed is in the range -255 to 255 with 0 being stop. Positive speed values are counter-clockwise rotation, and negative speed values are clockwise. The optional time parameter, if specified, has a valid range of 0.05 to 655.35 seconds. If not specified, a default of 5 seconds is used.
engines("ldir",left_val,"rdir",right_val,time)	Sets the left and right wheel to the specified speed levels, for an optional amount of time in seconds. The speed (left_val, right_val) values are in the range 0 to 255 with 0 being stop. The ldir and rdir parameters specify CW or CCW rotation of the respective wheels. The optional time parameter, if specified, has a valid range of 0.05 to 655.35 seconds. If not specified, a default of 5 seconds is used.

Path

Item	Description
waypoint_xythdrn()	Reads the x-coord, y-coord, time, heading, distance travelled, number of wheel revolutions, command number of the current waypoint. Returns a list with all these values as elements.
waypoint_prev	Reads the x-coord, y-coord, time, heading, distance travelled, number of wheel revolutions, command number of the previous waypoint.
waypoint_eta	Returns the estimated time to drive to a waypoint.
path_done()	Returns a value of 0 or 1 depending on whether the Rover is moving (0) or finished with all movement (1).
pathlist_x()	Returns a list of X values from the beginning to and including the current Waypoint X value.
pathlist_y()	Returns a list of Y values from the beginning to and including the current Waypoint Y value.
pathlist_time()	Returns a list of the time in seconds from the beginning to and including the current Waypoint time value.
pathlist_heading()	Returns a list of the headings from the beginning to and including the current Waypoint heading value.
pathlist_distance()	Returns a list of the distances travelled from the beginning to and including the current Waypoint distance value.
pathlist_revs()	Returns a list of the number of revolutions travelled from the beginning to and including the current Waypoint revolutions value.
pathlist_cmdnum()	Returns a list of command numbers for the path.
waypoint_x()	Returns x coordinate of current waypoint.
waypoint_y()	Returns y coordinate of current waypoint.
waypoint_time()	Returns time spent travelling from previous to current waypoint.
waypoint_heading()	Returns absolute heading of current waypoint.
waypoint_distance()	Returns distance travelled between previous and current waypoint.
waypoint_revs()	Returns number of revolutions needed to travel between previous and current waypoint.

Settings

Item	Description
units/s	Option for speed in grid units per second.
m/s	Option for speed in metres per second.
revs/s	Option for speed in wheel revolutions per second.
units	Option for distance in grid units.
m	Option for distance in metres.
revs	Option for distance in wheel revolutions.
degrees	Option for turning in degrees.
radians	Option for turning in radians.
gradians	Option for turning in gradians.
clockwise	Option for specifying wheel direction.
counter-clockwise	Option for specifying wheel direction.

Commands

These commands are collection of functions from other modules as well as from the TI Rover module.

Item	Description
sleep(seconds)	Pauses the program for the specified number of seconds. Imported from the time module.
text_at(row,"text","align")	Displays "text" in plotting area at specified "align". Imported from the ti_plotlib module.
cls()	Clears the Shell screen for plotting. Imported from the ti_plotlib module.
while get_key() != "esc":	Runs the commands in the "while" loop until the "esc" key is pressed.
wait_until_done()	Pauses the program until the Rover finishes the current command. This is a helpful way to synchronise non-Rover commands with Rover's motion.
while not path_done()	Runs the commands in the "while" loop until the Rover is finished with all movement. The path_done() function returns a value of 0 or 1 depending on whether the Rover is moving (0) or

Item	Description
	finished with all movement (1).
position(x,y)	Sets the Rover position on the virtual grid to the specified x,y coordinate.
position(x,y,heading,"unit")	Sets the Rover position on the virtual grid to the specified x,y coordinate, and the virtual heading, relative to the virtual x-axis is set if a heading is provided (in the units for angles specified). Positive angles from 0 to 360 are assumed to be counter-clockwise from the positive x axis. Negative angles from 0 to -360 are assumed to be clockwise from the positive x axis.
grid_origin()	Sets RV as being at current grid origin point of (0,0).
grid_m_unit(scale_value)	Sets the virtual grid spacing in metres per unit (m/unit) to the value specified. 0.1 is the default m/unit and translates to 1 unit = 100 mm or 10 cm or 1 dm or 0.1 m. The range of valid scale_value is from 0.01 to 10.0.
path_clear()	Clears any pre-existing path or waypoint information.
zero_gyro()	Resets the Rover gyro to 0.0 angle and clears the left and right wheel encoder counts.

Complex Maths Menu

This submenu is located under **More Modules**.

Item	Description
from cmath import*	Imports all methods from the cmath module.
complex(real,imag)	Returns a complex number.
rect(modulus,argument)	Converts polar coordinates to rectangular form of a complex number.
.real	Returns real part of the complex number.
.imag	Returns imaginary part of a complex number.
polar()	Converts rectangular form to polar coordinates of a complex number.
phase()	Returns phase of a complex number.
exp()	Returns e**x.
cos()	Returns cosine of a complex number.
sin()	Returns sine of a complex number.
log()	Returns natural logarithm of a complex number.
log10()	Returns base 10 logarithm of a complex number.
sqrt()	Returns square root of a complex number.

Time Menu

This submenu is located under **More Modules**.

Item	Description
from time import*	Imports all methods from the time module.
sleep(seconds)	Pauses the program for the specified number of seconds.
clock()	Returns the current processor time as a floating number expressed in seconds.
localtime()	Converts a time expressed in seconds since 1 January 2000 into a nine-tuple containing year, month, month day, hour, minute, second, weekday, year day, and Daylight Savings Time (DST) flag. If the optional (seconds) argument is not provided, then the real-time clock is used.
ticks_cpu()	Returns a processor specific increasing millisecond counter with arbitrary reference point. For measuring time consistently across different systems, use ticks_ms().
ticks_diff()	Measures period between consecutive calls to ticks_cpu() or ticks_ms(). This function should not be used to measure arbitrarily long periods of time.

TI System Menu

This submenu is located under **More Modules**.

Note: When creating a new program that uses this module, it is recommended to use the **Data Sharing** program type. This will ensure that all the relevant modules are imported.

Item	Description
from ti_system import*	Imports all methods (functions) from the ti_system module.
recall_value("name")	Recalls a predefined OS variable (value) named "name".
store_value("name",value)	Stores a Python variable (value) to an OS variable named "name".
recall_list("name")	Recalls a predefined OS list named "name".
store_list("name",list)	Stores a Python list (list) to an OS list variable named "name".
eval_function("name",value)	Evaluates a predefined OS function at the specified value.
get_platform()	Returns "hh" for handheld and "dt" for desktop.
get_key()	Returns a string representing the key pressed. The '1' key returns "1", 'esc' returns "esc", and so on. When called without any parameters - get_key() - it returns immediately. When called with a parameter - get_key(1) - it waits until a key is pressed.
get_mouse()	Returns mouse coordinates as a two element tuple, either the canvas pixel position or (-1,-1) if outside the canvas.
while get_key() != "esc":	Run the commands in the "while" loop until the "esc" key is pressed.
clear_history()	Clears the Shell history.
get_time_ms()	Returns time in milliseconds with millisecond precision. This functionality can be used to calculate a duration rather than determine the actual clock time.

TI Draw Menu

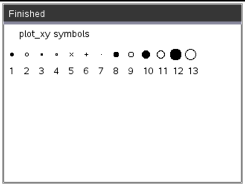
This submenu is located under **More Modules**.

Note: When creating a new program that uses this module, it is recommended to use the **Geometry Graphics** program type. This will ensure that all the relevant modules are imported.

Item	Description
from ti_draw import*	Imports all methods from the ti_draw module.

Shape

Item	Description
draw_line()	Draws a line starting from the specified x1,y1 coordinate to x2,y2.
draw_rect()	Draws a rectangle starting at the specified x,y coordinate with the specified width and height.
fill_rect()	Draws a rectangle starting at the specified x,y coordinate with the specified width and height and filled with the specified colour (using set_colour or black if not defined).
draw_circle()	Draws a circle starting at the specified x,y centre coordinate with the specified radius.
fill_circle()	Draws a circle starting at the specified x,y centre coordinate with the specified radius and filled with the specified colour (using set_colour or black if not defined).
draw_text()	Draws a text string starting at the specified x,y coordinate.
draw_arc()	Draws an arc starting at the specified x,y coordinate with the specified width, height and angles.
fill_arc()	Draws an arc starting at the specified x,y coordinate with the specified width, height and angles filled with the specified colour (using set_ colour or black if not defined).
draw_poly()	Draws a polygon using the specified x-list,y-list values.
fill_poly()	Draws a polygon using the specified x-list,y-list values filled with the specified colour (using set_colour or black if not defined).
plot_xy()	Draws a shape using the specified x,y coordinate and specified number from 1-13 representing different shapes and symbols (see below).

Item	Description
	 Finished plot_xy symbols 1 2 3 4 5 6 7 8 9 10 11 12 13

Control

Item	Description
clear()	Clears the entire screen. Can be used with x,y,width,height parameters to clear an existing rectangle.
clear_rect()	Clears the rectangle at the specified x,y coordinate with the specified width and height.
set_colour()	Sets the colour of the shape(s) that follow in the program until another colour is set.
set_pen()	Sets the specified thickness and style of the border when drawing shapes (not applicable when using fill commands).
set_window()	Sets the size of the window in which any shapes will be drawn. This function is useful to resize the window to match the data or to change the origin (0,0) of the drawing canvas.
get_screen_dim()	Returns the xmax and ymax of the screen dimensions.
use_buffer()	Enables an off-screen buffer to speed up drawing.
paint_buffer()	Displays the buffered drawing output. The use_buffer() and paint_buffer() functions are useful in cases where displaying multiple objects on the screen could cause delays.

Notes

- The default configuration has (0,0) in the top left corner of the screen. The positive x-axis points to the right and the positive y-axis points to the bottom This can be modified by using the set_window() function.
- The functions in ti_draw module are only available on the handheld and in handheld view on desktop.

TI Image Menu

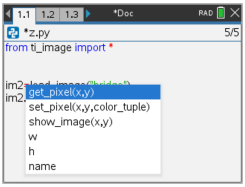
This submenu is located under **More Modules**.

Note: When creating a new program that uses this module, it is recommended to use the **Image Processing** program type. This will ensure that all the relevant modules are imported.

Item	Description
from ti_image import*	Imports all methods from the ti_image module.
new_image(width,height,(r,g,b))	Creates a new image with the specified width and height for use in the Python program. The colour of the new image is defined by the (r,g,b) values.
load_image("name")	Loads the image specified by the "name" for use in the Python program. The image must be part of the TNS document either in a Notes or Graphs application. The "name" prompt will display the image names (if they have been named earlier) or a number indicating their insertion order.
copy_image(image)	Creates a copy of the image specified by the "image" variable.

Methods of the image object

Additional functions related to the image objects are available in the Editor and Shell by typing the variable name followed by a. (dot).



```
from ti_image import *

im2=load_image("bridge")
im2.
```

- **get_pixel(x,y):** Gets the (r,g,b) value of the pixel at location defined by the (x,y) coordinate pair.

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

- **set_pixel(x,y,colour_tuple):** Sets the pixel at location (x,y) to the colour specified in the colour_tuple.

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

  Sets the pixel at (100,100) to the (0,0,255) colour.

- **show_image(x,y):** Displays the image with the top left corner at location (x,y).

- **w, h, name:** Gets the width, height, and name parameters of the image.

Example

```
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 Menu

Note: These lists do not include variables defined in any other TI-Nspire™ apps.

Item	Description
Vars: Current Program	(Editor only) Displays a list of global functions and variables defined in the current program
Vars: Last Run Program	(Shell only) Displays a list of global functions and variables defined in the last run program
Vars: All	(Shell only) Displays a list of global functions and variables from both the last run program and any imported modules

Appendix

Python Keywords

The following keywords are built into the TI-Nspire™ Python implementation.

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 Key Mapping

When entering code in the Editor or in the Shell, the keypad is designed to paste the appropriate Python operations or open menus for easy entry of functions, keywords, methods, operators, etc.

Key	Mapping
var	Opens Variables menu
sto→	Pastes = sign
del	Deletes character to the left of the cursor
clear	No action
=	Pastes = sign
|≠≥▸	Pastes the selected symbol(s): • >` • < • != • >= • <= • == • and • or • not • | • & • ~
trig	Pastes the selected function: • sin • cos • tan • atan2 • asin • acos • atan
?	Displays hints
:=	Pastes :=
^	Pastes **
$\sqrt[n]{x}$	No action
x^2	Pastes **2

Key	Mapping
√	Pastes sqrt()
×	Pastes multiply sign (*)
"□"	Pastes one double quote (")
÷	Pastes division sign (/)
$\frac{□}{□}$	No action
e^x	Pastes exp()
ln	Pastes log()
10^x	Pastes 10**
log	Pastes log(value,base)
(	Pastes (
)	Pastes)
[ ]	Pastes [ ]
{ }	Pastes { }
(–)	Pastes subtract sign (-)
≈	Adds a new line after the current line
EE	Pastes E
?!▸	Pastes the selected symbol(s): • ? • ! • $ • ° • ' • % • " • : • ; • _ • \\ • #
π▸	Pastes "pi"
⚐	Existing flag behaviour
↵	Adds a new line after the current line

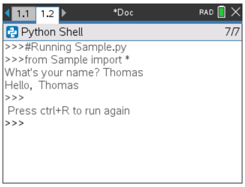
Sample Python Programs

Use the following sample programs to become familiar with Python methods. They are also available in the **Getting Started Python.tns** file located in the **Examples** folder.

Note: If you copy and paste any sample code that contains tab indent indicators (• •) to the TI-Nspire™ software, you will need to replace those instances with actual tab indents.

Hello

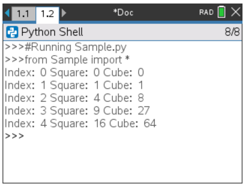
```
# This program asks for your name and uses
# it in an output message.
# Run the program here by typing "Ctrl R"

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



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

Loop Example

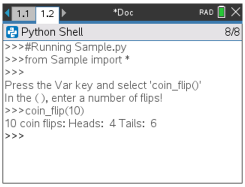
```
# This program uses a "for" loop to calculate
# the squares and cubes of the first 5 numbers
# 0,1,2,3,4
# Note: Python starts counting at 0

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



```
>>>#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

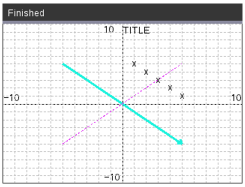
```
# Use random numbers to simulate a coin flip
# We will count the number of heads and tails
# Run the program here by typing "Ctrl R"

# 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!")
```



```
>>>#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
>>>
```

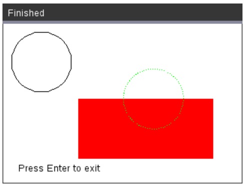
Plotting

```
# 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")
```



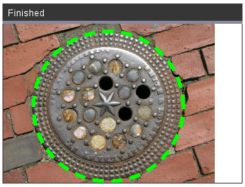
Drawing

```
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
im1=load_image("manhole_cover")
im1.show_image(0,0)
set_color(0,255,0)
set_pen("thick","dashed")
draw_circle(140,110,100)
```



Hub

This program uses Python to control the TI-Innovator™ Hub, a programmable microcontroller. Running the program without attaching a TI-Innovator™ Hub will display an error message.

For more information about TI-Innovator™ Hub, visit education.ti.com.

```
#========== 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

Select your country for more product information.

Contact TI Support

education.ti.com/ti-cares

Select your country for technical and other support resources.

Service and Warranty Information

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