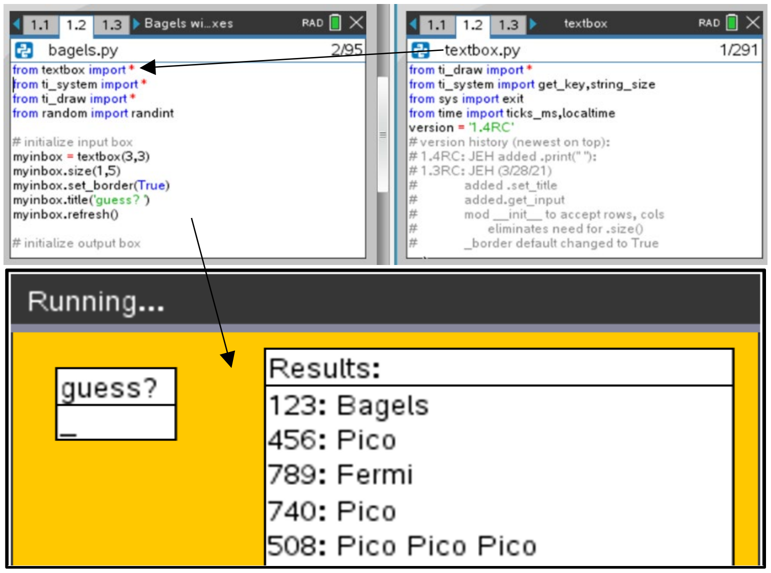
Do Python Modules Really Have More Fun? (TI-Nspire™ CX II)

Still John Hanna

Since Python programming was first released to the world on the **TI-84 Plus CE Python** and the **TI-Nspire CX II** about a year ago, I think it's time to dig a little deeper into the power of modular programming on these remarkable platforms. And discover why using Modules can be more fun, or can certainly at least save you some time.

Modules give me a powerful way to re-use common code. These custom support' files are easily shared among my many Python programs and can be shared with others.

Modular programming is also useful if you or your students want to work in teams: assign each member of the team a clearly defined task, have each develop a module, and merge the modules into a single TI-Nspire document.



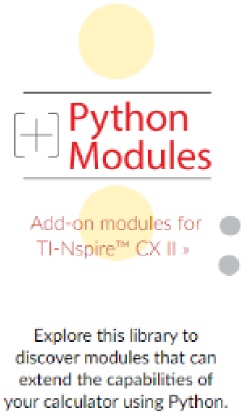
```
from textbox import *
from ti_system import *
from ti_draw import *
from random import randint

# initialize input box
myinbox = textbox(3,3)
myinbox.size(1,5)
myinbox.set_border(True)
myinbox.title('guess? ')
myinbox.refresh()

# initialize output box
```

```
from ti_draw import *
from ti_system import get_key,string_size
from sys import exit
from time import ticks_ms,localtime
version = '1.4RC'
# version history (newest on top):
# 1.4RC: JEH added .print(" "):
# 1.3RC: JEH (3/28/21)
#           added .set_title
#           added.get_input
#           mod __init__ to accept rows, cols
#               eliminates need for .size()
#           _border default changed to True
```

```
Running...
guess?
_
Results:
123: Bagels
456: Pico
789: Fermi
740: Pico
508: Pico Pico Pico
```

NEW: Python Modules Library on TI Codes

The developers at TI have included some special files in the **PyLib** folder: the modules beginning with **ti_***. If you download and install the very cool turtle module, it is installed into the **PyLib** folder as well. In fact, all Python files contained in the **PyLib** folder are available on the **[menu] > More Modules** list.


[+] Python Modules

Add-on modules for TI-Nspire™ CX II »

Explore this library to discover modules that can extend the capabilities of your calculator using Python.

The developers are busy creating other new modules to tap more of the potential of TI-Nspire through Python. For some practice with the TI modules see the additional **[+] Python Modules** section of TI Codes for Python TI-Nspire at https://education.ti.com/en/activities/ti-codes/python/ti-nspire-cx-ii.

Where are my files?

When I create a new TI-Nspire file using the TI-Nspire **Python Editor**, my ***.py** file resides *within* the document I'm working on (*duh*). But, unlike TI-Nspire 'core' variables that are created in the other Apps such as the Calculator, Lists & Spreadsheet, or even the TI-Basic Program Editor, the Python files are *not* restricted to just the current Problem, but are accessible in *all* Problems within the document. And those Python files can 'communicate' with each other.

What's a module?

I like to make a distinction between a Python **program** (also called a **script**) and a **module**. Both a program and a module are written using the same Python language but differ slightly in their *purpose*: a **program** is executable code that accomplishes an end goal: it should *do something*. A **module** is also a Python file, but it's end goal is *support*: it is designed to be imported into a program (or another module), like the **math** and **random** modules. A module is designed to make common routines 'portable' and 'reusable' and can contain any Python code, functions, and even class definitions.

Import this!

When I use an **import** statement in my program, I am invoking another Python file: a **module**. There are several modules included in the TI-Nspire Python system: some are standard **MicroPython** modules (**math**, **random**, **time**) and others are special TI-developed modules (**ti_system**, **ti_hub**, etc.).

There are three 'flavors' of **import** statements:

- **import math** - all functions must be preceded by the module name: y = **math**.sin(x)
- **import math as hoi** – all functions must be preceded by the 'alias' name you chose: y = **hoi**.sin(x)
- **from math import *** - all functions can be used without the module prefix: y = sin(x)

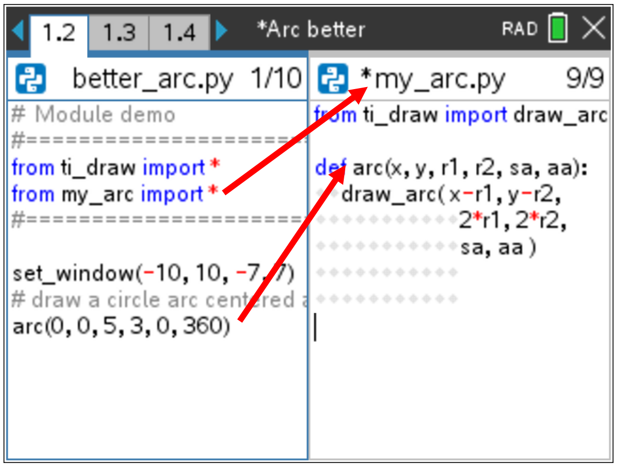
And, if you only intend to use one or two of the functions in a module you can just state which one(s) to import to conserve memory:

- **from random import randint, uniform** (*no parentheses after the function names*)

Writing a Module

The **draw_arc**( ) function in the **ti_draw** module can be tricky to use so I opted to create a custom **arc**( ) function to make it easier to position the arc…

In a *single* TI-Nspire document (***.TNS**), I can have *several* Python files (***.py**). For illustration purposes I set up a split screen page (below left) to show a complete program on the left side and the support **module** on the right side:

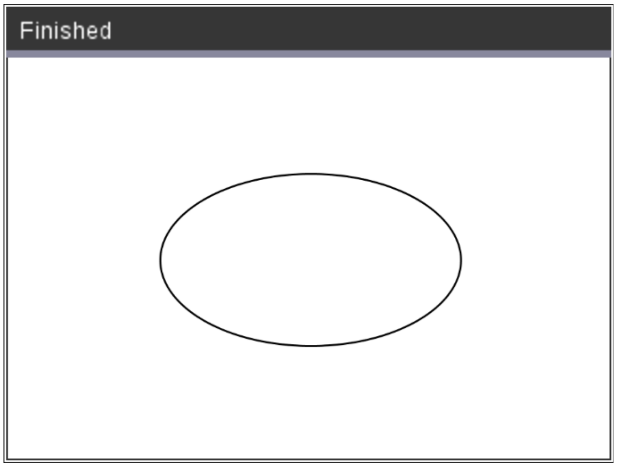


better_arc.py 1/10

```
# Module demo
#====================
from ti_draw import *
from my_arc import *
#====================

set_window(-10, 10, -7, 7)
# draw a circle arc centered a
arc(0, 0, 5, 3, 0, 360)
```

*my_arc.py 9/9

```
from ti_draw import draw_arc

def arc(x, y, r1, r2, sa, aa):
  draw_arc(x-r1, y-r2,
            2*r1, 2*r2,
            sa, aa )
```



program module output of program

The two Python files are:

- **better_arc.py** is a complete program (script) that imports the **my_arc.py** file and so has access to the **arc(…)** function. The program draws a complete ellipse centered at (0,0) with an x-radius of 5 and a y-radius of 3. The arguments 0, 360 ensure that the complete ellipse is drawn.

- **my_arc.py** (the module) contains just one function **def**inition: **arc(…)** that draws an *elliptical* arc using the center, x- and y-radii, start_angle and arc_angle. It must be stored (using ctrl-B or ctrl-R) before it can be used. There should *not* be an asterisk (*) before the **.py** filename at the top as seen above. (*Note: if you press* **ctrl-R** *in this file or* **import** *it into a Shell, nothing happens. But then press* **[var]** *and you will see the function* **arc()**. *You can use this function in the Shell but must provide values for the six arguments.*)

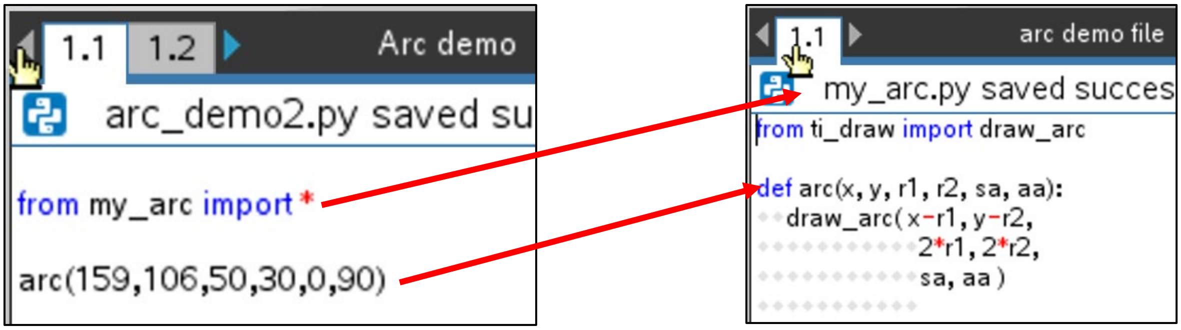
Did you notice that *both* Python files above import the **ti_draw** module? Python is designed to avoid duplication of functions and will not import those draw functions more than once. Whew.

Python modules can have lots of functions (and **class**es, too, but I'll save that topic for another post).

PyLib

A special folder in the TI-Nspire file system, **PyLib** is a place to store our Python modules so that they can be imported into any Python file in any TI-Nspire document on the device. If Python cannot locate an imported module *within* the current document (in *any* Problem), it then checks the **PyLib** folder to see if the module is located there. If not, then an error message is presented.

Thinking about that **arc( )** project from above: I create two *separate* *.tns files, one containing the program and another containing the module. The *.tns file containing the module (**my_arc.py**) is stored in the **Pylib** folder. Then any other python program in any other *.tns file can import the **my_arc** module to use that cool **arc(…)** function:



arc_demo2.py saved su

```
from my_arc import*

arc(159,106,50,30,0,90)
```

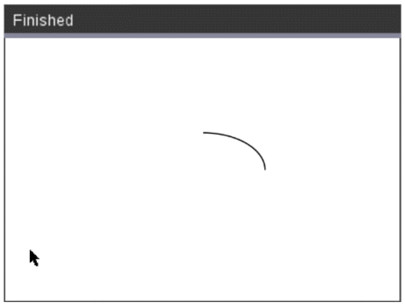
my_arc.py saved succes

```
from ti_draw import draw_arc

def arc(x, y, r1, r2, sa, aa):
  draw_arc(x-r1, y-r2,
           2*r1, 2*r2,
           sa, aa )
```

program: note the **import**

module in **PyLib**
note that TNS filename does not matter!

The output of **arc_demo2.py**:



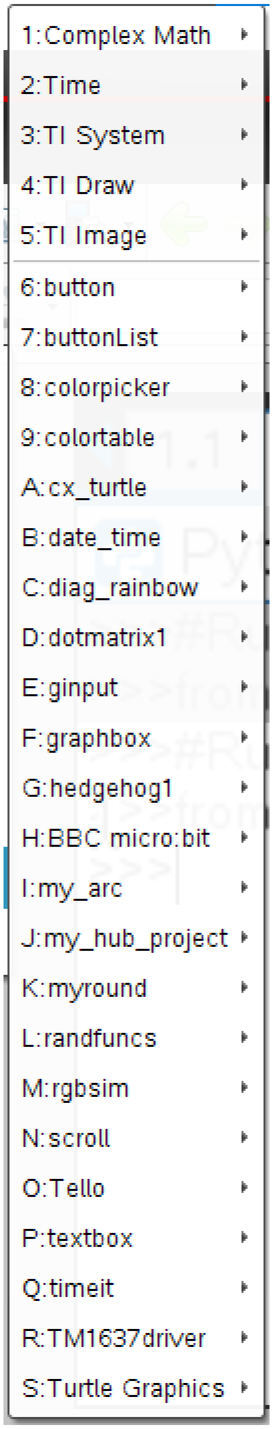
Note that **arc_demo2.py** file above does not import **ti_draw** and that **my_arc.py** only imports the **draw_arc** function from **ti_draw**. So all other **ti_draw** functions, like **set_window( )** are not available in either file. This program is thus using the *default* canvas: (0, 0)→(317, 211).

Note that Python does not care about *.tns filenames, only the *.py files stored within them.

My Own Pylib Folder

As you add modules to your **PyLib** folder they will appear on **[menu] > More Modules** in alphabetical order. *Note that the* ***.py** *files in the documents are listed, not the TI-Nspire documents*

To the right is the current list of my '**More Modules**' (snip taken from my computer screen). Above the separator bar are two standard modules and some TI-provided modules (1..5) that include robust sub-menus.



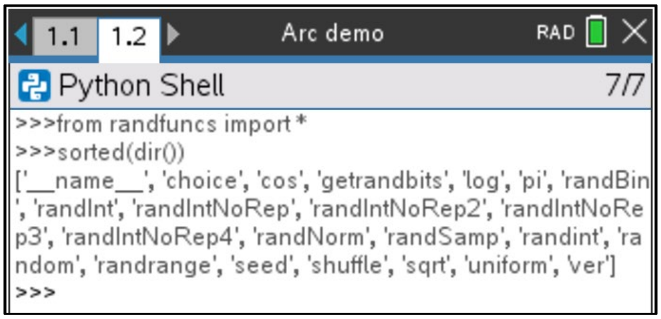
Below the separator bar are my own additional modules. Most of these sub-menus only contain an **import** statement but they are all in 'openable' TNS files so that I can read the documentation within the file (and the source code, too).

But there are (*so far*) three special modules on this list developed by TI that do contain submenus: the **BBC micro:bit**, **Tello**, and **Turtle Graphics** modules have menus from which to choose functions. The TI developers are really smart.

When developing a module to include in my **PyLib** folder, I *usually* give the TNS file <u>the same name</u> as the lone Python file contained within. This makes it easier to locate when I decide to open the file to read the docs or to edit the module. This was not the case with the **my_arc** module on purpose. But both **colorpicker** (a script) and **colortable** (a module) are in the same TNS document (for good reason, eh).

So, yes, a TNS file saved in the **PyLib** folder can contain *many* Python files and they will *all* appear separately on this list.

Python Tip: to see what functions are defined in a module: in any Shell, import the module then issue the command **sorted(dir())**:

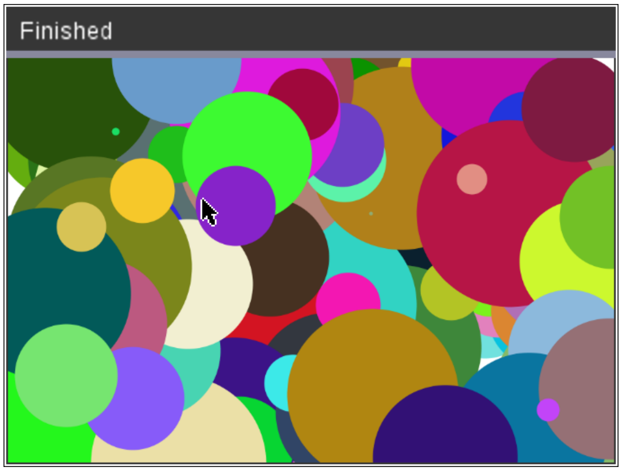


```
>>>from randfuncs import *
>>>sorted(dir())
['__name__', 'choice', 'cos', 'getrandbits', 'log', 'pi', 'randBin
', 'randInt', 'randIntNoRep', 'randIntNoRep2', 'randIntNoRe
p3', 'randIntNoRep4', 'randNorm', 'randSamp', 'randint', 'ra
ndom', 'randrange', 'seed', 'shuffle', 'sqrt', 'uniform', 'ver']
>>>
```

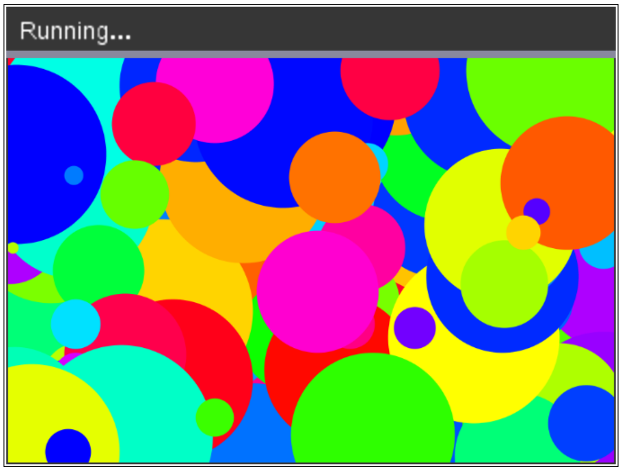
I get a listing of the function *names* only, but this is a start.
Remember that Python is Case Sensitive, so
randint( ) is different than **randInt( )**

My Favorite Module

Being a graphics fan, my favorite homemade module in my **Pylib** is **colortable**. This module contains a *list* of 360 colors of the rainbow. The list name is **Jcolors[ ]** and the index is in the range 0..359. Here's a sample of random circles made with plain, dull random colors and another made with vibrant random **Jcolors**:

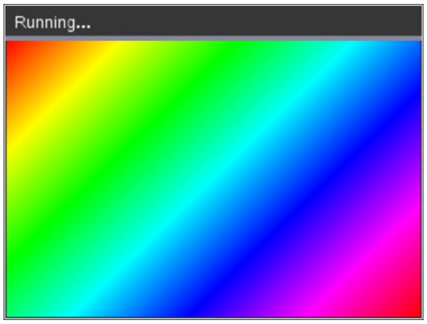


```
r, g, b=randint(0, 255), randint(0, 255),
        randint(0, 255)
set_color(r, g, b)
```



```
from colortable import *

set_color( Jcolors[ randint(0, 359) ] )
```

And here's a screenshot of the 360 colors:



```
from ti_draw import *
from colortable import *
for i in range(360):
  set_pen(1, 0)
  set_color(Jcolors[i])
  draw_line(1.5*i, -1, -1, 1.46*i)
```

Summary

Python modules give me a powerful way to re-use common code. These modules or 'support' files are easily shared among my many Python programs and can be shared with others. I'm looking forward to your contributions to the TI-Nspire Python community. And special thanks to the developers at Texas Instruments for creating additional powerful tools for a rich and engaging Python experience.