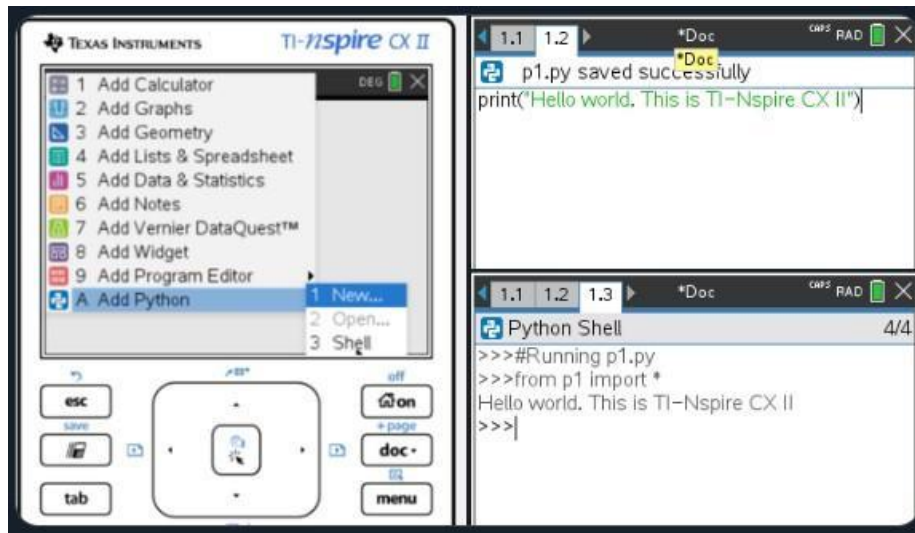


CX II does Python!

John Hanna, johnhanna@gmail.com

Steve Phelps, sphelps31415@gmail.com

Prgmers rule the world.



Overview: Included in CX II OS version 5.2.xxx

Why Python?

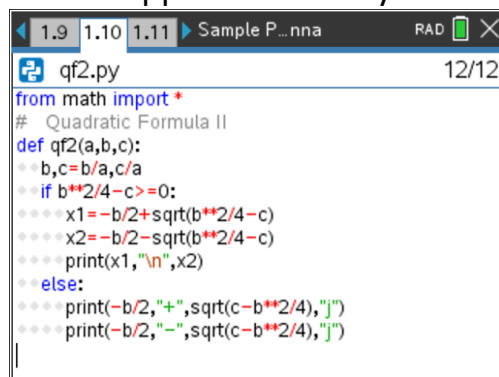
- popular, fast, and powerful
- *easy* to learn
- Procedural, Object-Oriented *and* Functional
- robust - mature (>30 years)
- expandable (Libraries/Packages galore)
- many platforms
- “low floor, high ceiling”
- “batteries included”

<https://spectrum.ieee.org/at-work/tech-careers/top-programming-language-2020>

TI-NsPython

There is a new item on the 'New Page' menu: **A: Add Python**

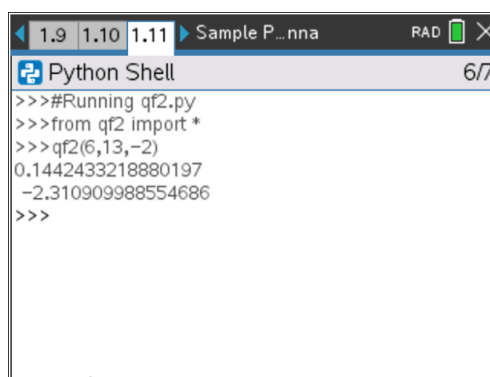
There are **Two** new apps in the family:



The screenshot shows the TI-NsPython Editor window. The title bar includes version numbers 1.9, 1.10, and 1.11, and a file named 'Sample P...nna'. The editor displays a Python script named 'qf2.py' with 12 lines of code. The code defines a function 'qf2(a,b,c)' that calculates the roots of a quadratic equation. It includes comments, imports, and conditional logic to handle different cases of the discriminant.

```
from math import *
# Quadratic Formula II
def qf2(a,b,c):
    b,c=b/a,c/a
    if b**2/4-c>=0:
        x1=-b/2+sqrt(b**2/4-c)
        x2=-b/2-sqrt(b**2/4-c)
        print(x1,"\\n",x2)
    else:
        print(-b/2,"+",sqrt(c-b**2/4),"j")
        print(-b/2,"-",sqrt(c-b**2/4),"j")
```

Editor



The screenshot shows the TI-NsPython Python Shell window. The title bar includes version numbers 1.9, 1.10, and 1.11, and a file named 'Sample P...nna'. The shell displays the output of running the 'qf2.py' script. It shows the function being imported and the results of calling 'qf2(6,13,-2)', which are two complex numbers.

```
>>>#Running qf2.py
>>>from qf2 import *
>>>qf2(6,13,-2)
0.1442433218880197
-2.310909988554686
>>>
```

Shell

The Python Experience on TI-Nspire CX II

- Designed for Middle and High School Mathematics and Science classes
- Robust set of menu items for easy 'pick-from-the-menu' programming
- Special inline prompts and tooltips when selecting commands from the menu
- MicroPython modules: math, random, cmath, time, and TI-specific modules for graphics, TI-Innovator/Rover, system
- Familiar TI-Basic shortcuts (ctrl-T, ctrl-B, ctrl-R, ...)
- Beautiful, **colorful Editor**
- **Shell**: like the Calculator app; can test expressions and write code chunks
- **Editor** and **Shell** both have special key mappings (see ctrl=, Var).
- Full TI-Nspire integration including 'physical computing' with *simple* Innovator and Rover coding using an *Object-Oriented* approach

New to Python?

Resources:

www.python.org - The home of Python

<https://docs.micropython.org/> - MicroPython documentation

www.w3schools.com/python/ - Python lessons

<https://trinket.io/> - online coding platform & lessons

<https://www.pythonforbeginners.com/> - more lessons

<https://codingbat.com/python> - exercises checked online

[https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)) - info

New: TI Codes CX Python course: [TI Codes](#)

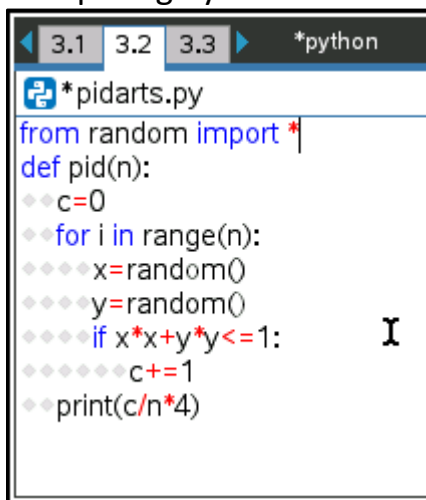
CX Python Innovator/Rover 'coming soon'

- The **Editor** is for writing Python code
- Use the **[menu]** items for inline prompts, tooltips and proper syntax
- Python is CaSe SeNsItIvE! *not* If *but* if
- Use == ("equals") for conditions (as in **if a == b:**); use = ("gets") for assignment (as in **a = a+1**)
- Indentation (the "off-side rule"): *(note the colon)*
determines blocks (**if**, **while**, **for**, **def**, etc.). The Editor assists
- The **Shell** is the Python 'run' environment. >>> is the Shell "command prompt"
- Have a question? – 'google' it!

Experienced with Python?

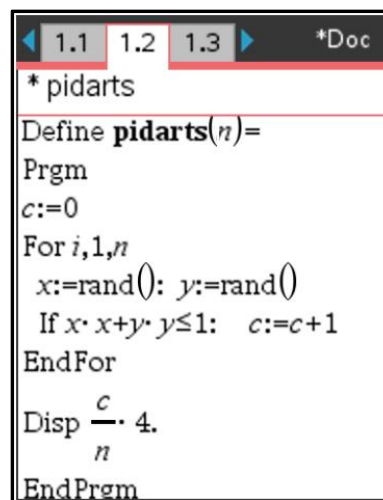
- **Editor**: use the [menu] for easy code entry, including indent
- **Shell**: see **[menu] > Rerun last program (ctrl-R)** (unique feature)
- Can add modules that are compatible with CXII OS; can write modules
- New **Pylib** folder for 'public' modules
- See 'Under the Hood' later in this doc

For Beginners: Comparing Python and TI-Basic:



```
*pidarts.py
from random import *
def pid(n):
    c=0
    for i in range(n):
        x=random()
        y=random()
        if x*x+y*y<=1:
            c+=1
    print(c/n*4)
```

Python



```
*pidarts
Define pidarts(n)=
Prgm
c:=0
For i,1,n
  x:=rand(): y:=rand()
  If x*x+y*y<=1: c:=c+1
EndFor
Disp c/n^4.
EndPrgm
```

TI-Basic

Your first Python program (for those needing step-by-step):

[home] > New document

Add Python > New... type **hello** for the program name [enter]

first line of code:

type **name=**

[menu] > Built-ins > I/O > input()

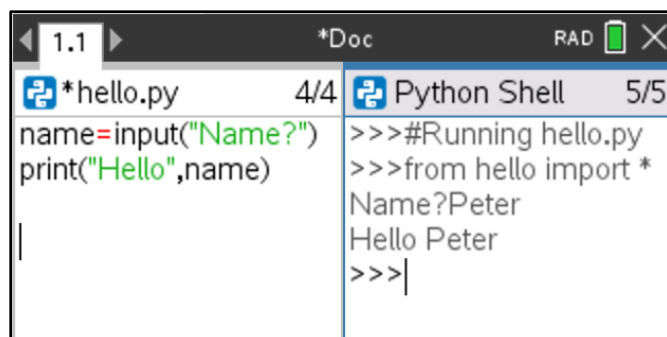
inside the (), type **"Name?"** after the) press [enter]

second line of code:

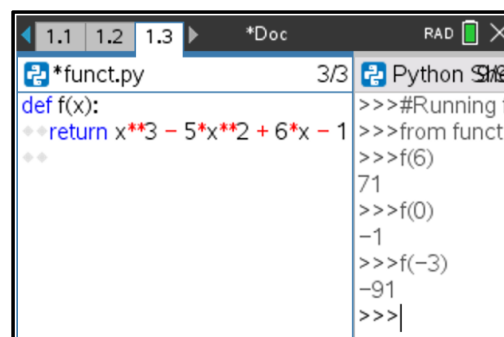
[menu] > Built-ins > I/O > print()

inside the parens type **"Hello", name**

press **ctrl-r** to run..... then try this:



```
*hello.py 4/4 Python Shell 5/5
name=input("Name?")
print("Hello",name)
>>>#Running hello.py
>>>from hello import *
Name?Peter
Hello Peter
>>>
```

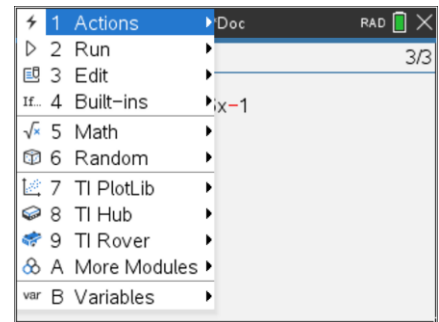


```
*funct.py 3/3 Python Shell 5/5
def f(x):
    return x**3 - 5*x**2 + 6*x - 1
>>>#Running f
>>>from funct i
>>>f(6)
71
>>>f(0)
-1
>>>f(-3)
-91
>>>
```

(split screens are for demo purposes showing code and result of execution)

Python Editor [menu]

Not all available modules/functions are on the **Built-ins**, **Math**, and **Random** menus. The most common tools programming are available.



Python Shell [menu]

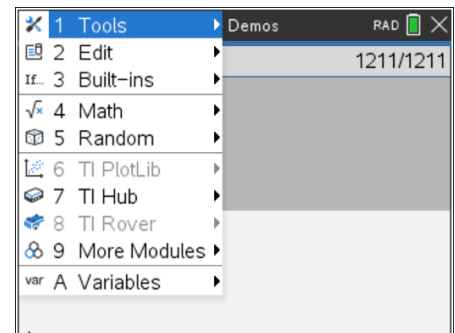
'Run...' lets you select the filename to run. Use this feature if/when **ctrl-r** fails (it's Python's fault!)

dir() returns the list of *names* in the current scope (Shell)

[var] shows defined identifiers (vars & functions)

Coding menus duplicated here for trying things out.

From PROGRAM import * lets you select a Python file to import



MicroPython documentation

<https://docs.micropython.org/en/latest/index.html>

MicroPython built-ins documentation can be found at

<https://docs.micropython.org/en/latest/library/builtins.html>

but not all MicroPython modules are included in CX II Python.

TI Custom Modules

ti_plotlib - facilitates quick graphing and graphical display of data

ti_hub - control TI-Innovator Hub

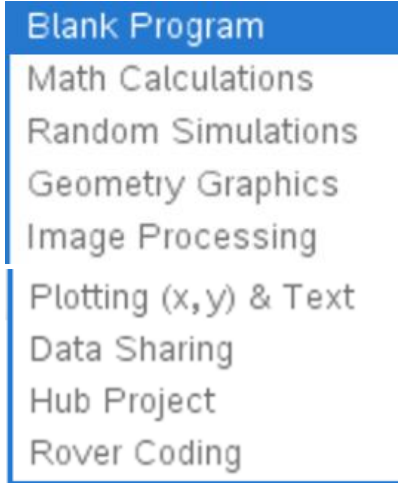
ti_rover - control TI-Innovator Rover

ti_system - interoperability with TI-Nspire native functions and variables

ti_draw - draw geometric graphics (like Draw in TI-Basic)

ti_image - image processing (manipulation) also good for animation

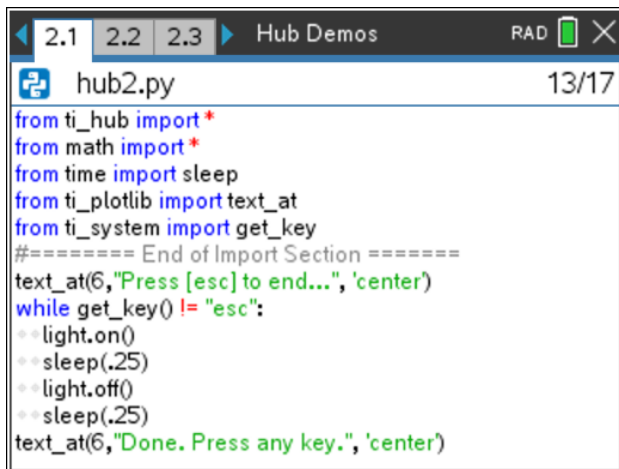
TI-NsPython Templates



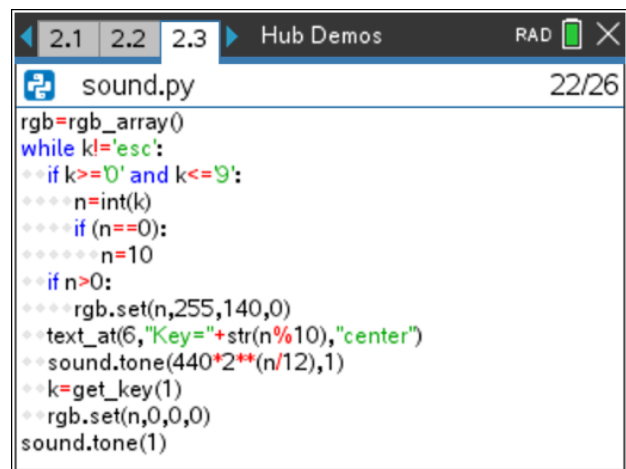
Most templates provide *only* the **import** statements for *commonly* used tools for that type of programming project. Other things can be imported by the programmer.

The Hub and Rover templates check to make sure that the TI-Innovator Hub is attached. If not, the program halts. But... there is a workaround.

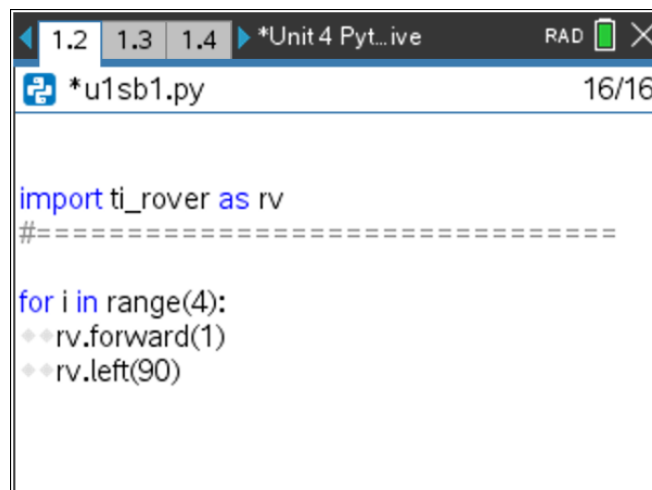
Python and the TI-Innovator Hub



```
from ti_hub import *
from math import *
from time import sleep
from ti_plotlib import text_at
from ti_system import get_key
#===== End of Import Section =====
text_at(6,"Press [esc] to end...", 'center')
while get_key() != "esc":
    light.on()
    sleep(.25)
    light.off()
    sleep(.25)
text_at(6,"Done. Press any key.", 'center')
```



```
rgb=rgb_array()
while k!='esc':
    if k>='0' and k<='9':
        n=int(k)
        if (n==0):
            n=10
        if n>0:
            rgb.set(n,255,140,0)
            text_at(6,"Key="+str(n%10),"center")
            sound.tone(440*2**(n/12),1)
            k=get_key(1)
            rgb.set(n,0,0,0)
            sound.tone(1)
```



```
import ti_rover as rv
#=====

for i in range(4):
    rv.forward(1)
    rv.left(90)
```

Under the Hood: Python Scope in a TI-Nspire Document

- Python programs can run in the Python Shell *only*.
- Python variables can be linked to TI-Nspire variables via **store_var()** and **recall_var()** functions found in **ti_system**.
- All Python files in a document are available to open *anywhere* in the document (use Python>Open or Shell>from PROGRAM import *)
- There is only one Shell *environment* per Problem (but with separate text histories). (See demo Canada flags.tns)
- Python programs, functions & vars are *not* TI-Nspire vars.
- Pressing [var] in Editor or Shell exposes only Python vars & functions and, in Editor, cursor position affects visible vars
- Can copy/paste Python code from other sources into/out of Premium software.
- Shell history is *not* saved with the document. Next time you open the document all Shells are there but are empty except for the Shell prompt '>>>'.
- Python *.py files are stored within the document.
- There is a new PyLib folder (similar to MyLib).

