

Using TI-Innovator Hub under Python

Veit Berger, Hans-Martin Hilbig



Teachers Teaching with Technology™

t3europe.eu

Agenda

- Concepts of Object-Oriented Programming from an educator's perspective
- Adding new sensors under Python
- Using "time" functions on TI-Nspire CXII and TI-Nspire Desktop

Before you go:

- Please give us detailed feedback about what you liked / disliked
- Please give us detailed feedback about where you need help!



TI-Nspire Python from the educator's perspective

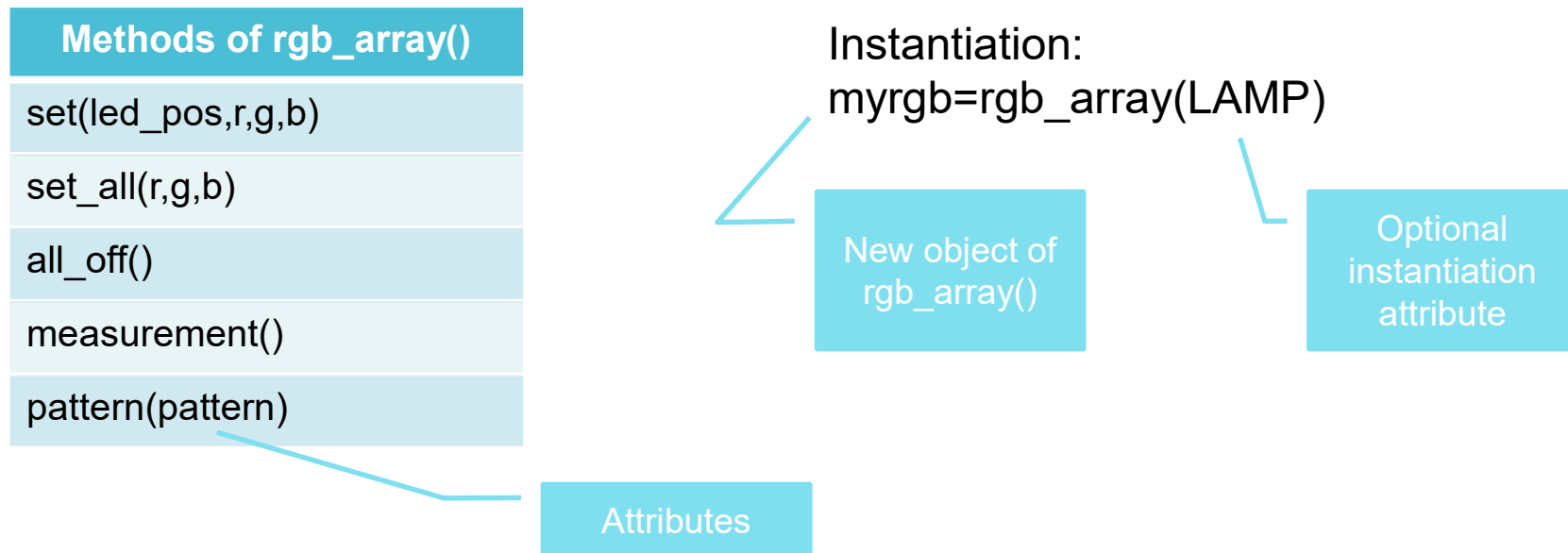
Veit Berger,
Teacher Computer Science & Physics, Geschwister-Scholl-Gymnasium
Löbau:

1. MicroPython, as implemented in the TI-Nspire-CXII, satisfies all needs of an Object-Oriented Computer Science education
2. Highly abstract Object Libraries will enable students to create attractive applications based on simple code



Ref 1: Object-Oriented Programming (1)

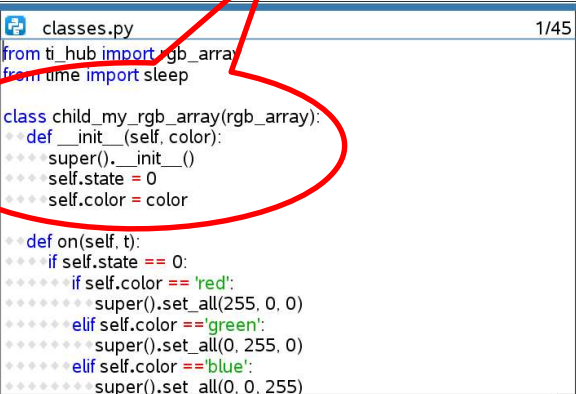
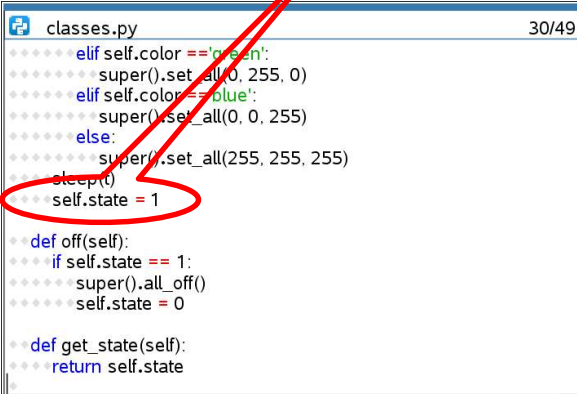
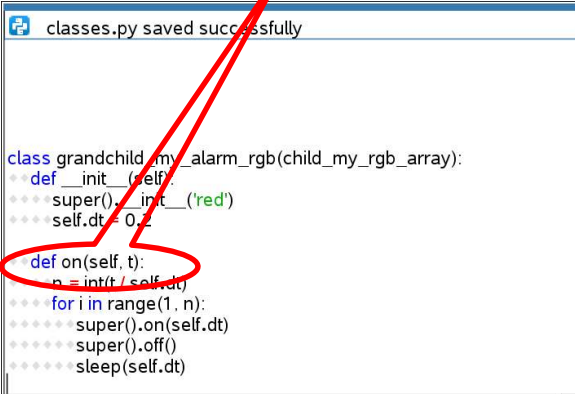
Example: The `rgb_array()` class (as part of the `ti_hub` module)



Ref 1: Object-Oriented Programming (2)

Extending of what's available

Inheritance	Encapsulation	Polymorphism
Adding a new class with inheritance of methods and properties from the parent class	Internal class variables are not accessible from outside the object	At runtime it is decided which method of the same name of a class hierarchy is executed

		
--	---	--



Ref 2: Highly abstract Object Libraries (1)

“Space Hedgehog” (Raumigel), a highly abstract 3D-graphics library



Allows creation of 3D objects in a coordinate-free Graphics Space, similar to the popular 2D Python turtlegraphics library

- First created* in LOGO by Lötke, Wölpert, Wolpert; Raumigel - Einführung, Anwendungen, Implementation; Informatik und Datenverarbeitung in der Schule, Materialien und Berichte Nr. 7, Pädagogische Hochschule Ludwigsburg, 1985
- Migrated to Pascal by Veit Berger, 1995
- Migrated to Delphi by Veit Berger, 2000
- Migrated to TI-Nspire MicroPython by Veit Berger, 2020



Ref 2: Highly abstract Object Libraries (2)

Create compact code, based on a highly abstract 3D-graphics library

Coordinate-free graphics	Train imagination in 3D	Advanced applications
Grades 8-9: experiment – reflect – learn	Grades 9-10: Learn Math & Geometry	Grades 10+: Applied trigonometry

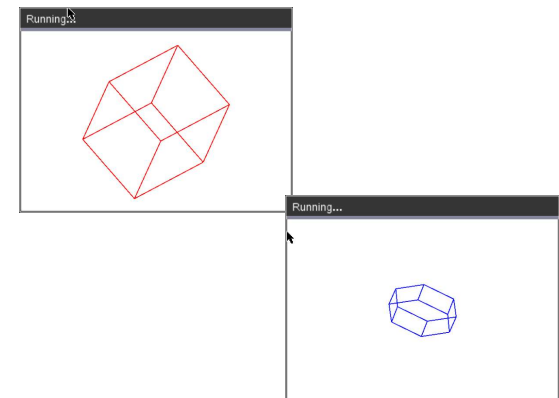
```
igel2.py 17/49
from classes import *
from ti_system import get_key

def draw(obj, dist):
    for i in range(4):
        for j in range(4):
            obj.vw(dist)
            obj.li(90)
        obj.pen_up()
        obj.vw(dist)
        obj.pen_down()
    igel.kvo(90)
```

```
prisma1.py 1/60
from classes import *

def draw(obj, l, h, n):
    angle = 360 / n
    for i in range(n):
        for j in range(2):
            obj.vw(l)
            obj.li(90)
            obj.vw(h)
            obj.li(90)
        obj.vw(l)
    obj.kvo(angle)

def main():
    input = screen("Regelmäßige N-eckige Prismen", "Dateneingabe")
    length = input.set_value(3, 1, "Seitenlänge in Pixeln: ")
    height = input.set_value(4, 1, "Höhe in Pixeln: ")
```



Teachers Teaching with Technology™

VB/HMH 10/15/20

t3europe.eu

Ref 2: Highly abstract Object Libraries (3)

Adding a new accelerometer sensor library (ADXL335)

Lowest level, simple	Intermediate level	Highest level, complex
Hardware-level readout of 3-axis ADC values	Offset & gain compensation, transform into a physics property (gravitational acceleration)	Calculation of 3D-Euler angles, transform into a mathematical model

```

ADXL335.py 47/89
self._dx=analog_in(ax)
self._dy=analog_in(ay)
self._dz=analog_in(az)
if avg!=0:
    if avg==0:
        avg=1
    if avg>25:
        avg=25
    send("set averaging "+str(avg)+"")
def get_adcxyz(self):
    self._xyz = []
    self._xyz += [int(self._dx.measurement())]
    self._xyz += [int(self._dy.measurement())]
    self._xyz += [int(self._dz.measurement())]
    return self._xyz
    
```

```

ADXL335.py 65/99
def get_compadcxyz(self): #compensated ADC reads
    self._compxyz=[]
    self._xyz=self.get_adcxyz()
    self._compxyz+=[round(((self._xyz[0]-self.off_x)/self.gain_x),2)]
    self._compxyz+=[round(((self._xyz[1]-self.off_y)/self.gain_y),2)]
    self._compxyz+=[round(((self._xyz[2]-self.off_z)/self.gain_z),2)]
    return self._compxyz
    
```

```

ADXL335.py 82/99
def get_anglexyz(self): #Euler angles
    self._anglexyz = []
    self._xyz=self.get_compadcxyz()
    self._divx=sqrt(self._xyz[1]**2+self._xyz[2]**2)
    if self._divx!=0:
        self._theta=degrees(atan(self._xyz[0]/self._divx))
        if self._xyz[1]>0 and self._xyz[2]<0:
            if self._xyz[0]>0:
                self._theta=180-self._theta
            else:
                self._theta=-self._theta
    else:
        self._theta=90.0
    self._divy=sqrt(self._xyz[0]**2+self._xyz[2]**2)
    if self._divy!=0:
        self._psi=degrees(atan(self._xyz[1]/self._divy))
        if self._xyz[0]>0 and self._xyz[2]<0:
    
```



The TI-Nspire Python 'time' module (1)

```

1.1 1.2 1.3 raumigeB...xyz RAD
Python-Shell 6/6
>>>from time import *
>>>sorted(dir())
['__name__', 'clock', 'localtime', 'sleep', 'sleep_
ms', 'sleep_us', 'ticks_add', 'ticks_cpu', 'ticks_dif
f', 'ticks_ms', 'ticks_us', 'time']
>>>|

```

Legacy time functions	Ticks-counter based time functions	Real-Time-Clock(RTC) based time functions
wait()	sleep()	clock()
hub_time()	sleep_ms()	time()
	sleep_us()	localtime()
	ticks_cpu()	
	ticks_ms()	
	ticks_us()	
	ticks_diff()	
	ticks_add()	



The TI-Nspire Python 'time' module (2)

Differences between Handheld and Desktop:

Function	Handheld	Desktop	Rollover	
			Handheld	Desktop
ticks_cpu()	30-bit integer, 10ms LSB	62-bit integer, 100ns LSB	124 days*	>>years***
ticks_ms()	30-bit integer, 10ms LSB	62-bit integer, 1ms LSB	12 days*	>>years***
ticks_us()	30-bit integer, 10ms LSB	62-bit integer, 1us LSB	17 mins*	>>years***
clock()	Float, 10ms resolution	Float, 1ms resolution	Note *	>>years***
time()	Integer, 1s resolution	Float, 10us resolution	Note **	N/A
localtime()	Tuple, synced from Desktop	Tuple, synced from Server	Note **	N/A

Notes: * counters are paused when Handheld is in Standby, ** data is cleared when Handheld is in Hibernate, *** counters appear to be random after Desktop App started, so Rollover might happen sooner!



Using 'time' for benchmarks

```
benchmark.py 15/15
from time import *

tloop=i=0
t0=ticks_ms() #store current timer value

while ticks_diff(ticks_ms(),t0)<1000: # run code in a loop for 1sec
    t1=ticks_ms() # this is optional
    i+=1
    print("Loop: ",i," time: ",tloop) #code to benchmark goes here
    tloop=ticks_diff(ticks_ms(),t1) # this is optional

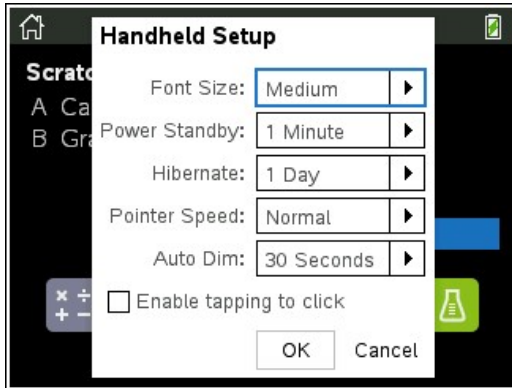
print("total execution time [ms]: ",ticks_diff(ticks_ms(),t0))
print("total # of loops: ",i)
print("time in loop: [ms] ",tloop)
```

- Use ticks_ms() to measure code execution
- Place your code to benchmark in a 1 second loop
- Optionally, measure execution time of each loop
- Measure execution time right after loop
- Use ticks_diff(b,a) instead of (b-a), to avoid rollover error

- Start small, to identify the time-consuming parts of your code
- Be careful when you compare benchmarks across platforms:
 - CPU speed and OS (Win10/MacOS) of your PC/Notebook matters
 - Apps (VPN, Browser, Excel, etc.) open in parallel eat CPU time
 - Handheld performs differently to PC



TI-Nspire-CXII Standby & Hibernate



	Standby	Hibernate
ticks_counters	Paused	Reset
Time,localtime	Continued	Cleared
Python Shell	Saved	Reinitialized

- Make sure all Handhelds of the classroom use the same settings
- Synchronize Real Time Clock of Handheld after Hibernate, by connecting Handheld to Nspire Desktop
- To avoid loss of RTC data, set Hibernate to <never>, at the expense of battery lifetime

The TI-Nspire Python 'time' module summary

- Use of legacy functions (`hub_time()`,`wait()`) is not recommended
- Time functions behave differently on Handheld vs. Desktop
- Handheld uses 30-bit wide, Desktop uses 62-bit wide counters
- `ticks_ms()` works best for TI-Nspire Python code benchmarking
- Use `ticks_diff()` when benchmarking code, to avoid rollover errors
- `ticks_cpu()` provides highest resolution
- Maximum time resolution of Handheld is 10 milliseconds
- Maximum time resolution of Desktop is 100 nanoseconds
- After Hibernate mode, all handheld timers are reset!
- Regularly plug Handheld to Desktop to update `time()`,`localtime()`!



Helpful links

- TI Education Technology Homepage, Documentation
<https://education.ti.com/>
<https://education.ti.com/en/guidebook/search/ti-nspire-cx>
- T³ Europe – Webinars live and on-demand, Documentation, Material Database
<https://www.t3europe.eu/en/t3-europe/webinars/on-demand>
<https://ti-unterrichtsmaterialien.net/materialien>
- MicroPython documentation
<https://docs.micropython.org/en/latest/index.html>
- MicroPython Online-reference with tutorials
<https://www.programiz.com/python-programming/first-program>
- Gymnasium Löbau – computer science homepage
<https://gsg-loebau.de/fachbereiche/naturwissenschaften/informatik>
- ... or simply shoot us an email with questions, suggestions, feedback to:
hm-hilbig@web.de v.berger@gmx.de

