
```
In [1]: from astroquery.simbad import Simbad
```

```
In [2]: Simbad.add_votable_fields("flux(V)","flux(U)","flux(B)")
```

```
In [3]: obj = Simbad.query_object("aldebaran")
```

```
In [4]: obj.pprint()
```

MAIN_ID	RA	DEC	RA_PREC	DEC_PREC	...	COO_BIBCODE	FLUX_V
FLUX_U	FLUX_B						
	"h:m:s"	"d:m:s"			...		mag
mag	mag						

* alf Tau 04 35 55.2390	+16 30 33.488	9	9	...	2007A&A...474..653V	0.860000	

```
In [5]: type(obj)
```

```
Out[5]: astropy.table.table.Table
```

```
In [6]: obj.keys()
```

```
Out[6]:
```

```
['MAIN_ID',  
 'RA',  
 'DEC',  
 'RA_PREC',  
 'DEC_PREC',  
 'COO_ERR_MAJA',  
 'COO_ERR_MINA',  
 'COO_ERR_ANGLE',  
 'COO_QUAL',  
 'COO_WAVELENGTH',  
 'COO_BIBCODE',  
 'FLUX_V',  
 'FLUX_U',  
 'FLUX_B']
```

```
In [7]: obj["FLUX_V"]
```

```
Out[7]:
```

```
<MaskedColumn name='FLUX_V' dtype='float32' unit='mag' format=u'!r:>'  
0.86000001
```

- Per un progetto di natura divulgativa, devo costruire una lista di stelle “note”: ovvero con un nome registrato, visibili, appartenenti a costellazioni ben conosciute.
- Per quelle principali, vorrei calcolare il periodo di massima visibilità

Procedimento:

- 1 Consultando qualche sito web¹, con un po' di editing, ho ricavato una lista di nomi che ho trascritto nel file `starlist.py`.

file: `starlist.py`

Elenco stelle "con nome" e relativo nome ufficiale.

```
#           Nome tradizionale           Nome canonico
STAR_DB = [["Acamar",                "Theta 1 Eridani"],
            ["Achernar",              "Alpha Eridani"],
            ["Achird",                 "Eta Cassiopeiae"],
            ["Acrab",                  "Beta Scorpii"],
            ...,
            ["Zubenelgenubi",          "Alpha 2 Librae"],
            ["Zubeneschamali",         "Beta Librae"]]
```

1) Ad esempio:

http://www.astro.wisc.edu/~dolan/constellations/starname_list.html
<https://www.naic.edu/~gibson/starnames/starnames.html>
<http://www.darkshire.net/jhkim/rpg/polaris/starnames.html>
https://en.wikipedia.org/wiki/List_of_proper_names_of_stars

- ② Per i passi successivi ho più volte utilizzato il modulo Simbad del package astroquery per:
- verificare l'esistenza del nome
 - ricavare i parametri (incluso il nome “canonico”)

file: `starinfo.py`

```
import time
import sys

from astroquery.simbad import Simbad

from starlist import STAR_DB

Simbad.add_votable_fields("flux(U)", "flux(B)", "flux(V)")

if len(sys.argv) > 1:
    first = sys.argv[1][0].upper()
else:
    first = None

for star in STAR_DB:
    if first and star[0][0].upper() != first:
        continue
    time.sleep(1)
    try:
        obj = Simbad.query_object(star[0])
    except Exception as e:
        print "Errore:", str(e)
    else:
        if obj:
            obj_data = [obj["MAIN_ID"].data[0], obj["RA"].data[0],
                        obj["DEC"].data[0], obj["FLUX_U"].data[0],
                        obj["FLUX_B"].data[0], obj["FLUX_V"].data[0]]
            star.extend(obj_data)
            print star
        else:
            print "%s (%s): non trovata" % (star[0], star[1])
```

Esecuzione di `starinfo.py`

```
In [1]: %run starinfo.py
['Acamar', 'Theta 1 Eridani', '* tet01 Eri', u'02 58 15.696', u'-40 18 16.97', mas
['Achernar', 'Alpha Eridani', '* alf Eri', u'01 37 42.8454', u'-57 14 12.310', -0.
['Achird', 'Eta Cassiopeiae', '* eta Cas', u'00 49 06.2907', u'+57 48 54.675', 4.0
['Acrab', 'Beta Scorpii', '* bet Sco', u'16 05 26.23', u'-19 48 19.4', 2.0, 2.4300
['Acrux', 'Alpha Crucis', '* alf01 Cru', u'12 26 35.871', u'-63 05 56.58', masked,
....
Al Dhanab (Gamma Gruis): non trovata
....
['Zosma', 'Delta Leonis', '* del Leo', u'11 14 06.5014', u'+20 31 25.385', 2.79, 2
['Zubenelgenubi', 'Alpha 2 Librae', '* alf02 Lib', u'14 50 52.7130', u'-16 02 30.3
['Zubeneschamali', 'Beta Librae', '* bet Lib', u'15 17 00.4138', u'-09 22 58.491',
```

Note:

- Notare l'istruzione `time.sleep(1)`.
- Per tutti gli oggetti non trovati (es: Al Dhanab) ho effettuato qualche ricerca manuale fino a determinare:
 - Nome non esistente
 - Nome con diversa grafia
 - Errore nell'accesso al database Simbad

Modificando poi manualmente il file `starlist.py`

- Dal database Simbad ho ricavato il nome canonico della stella, le coordinate, i flussi U, B, V

3 Il caso dell'errore in Simbad:

```
In [4]: aldanab = Simbad.query_object("al dhanab")
/usr/local/lib/python2.7/dist-packages/astroquery/simbad/core.py:136: UserWarning:
  (error.line, error.msg))
```

```
In [5]: aldanab = Simbad.query_object("gamma gruis")
```

```
In [6]: aldanab
```

```
Out[6]:
```

```
<Table masked=True length=1>
```

MAIN_ID	RA	DEC	RA_PREC	DEC_PREC	COO_ERR_MAJA	...	FLUX_B	FLUX_V
FLUX_B_1	FLUX_V_1							
	"h:m:s"	"d:m:s"			mas	...	mag	mag
mag	mag	mag						
object	unicode13	unicode13	int16	int16	float32	...	float32	float32
float32	float32							

* gam Gru	21 53 55.7262	-37 21 53.479	9	9	5.280	...	2.8900001	3.01
2.52	2.8900001	3.01						

```
In [7]: Simbad.query_objectids("gam gru")
```

```
Out[7]:
```

```
<Table length=32>
```

```
ID
str23
```

```
-----
NAME Al Dhanab
PLX 5287
* gam Gru
CD-37 14536
```

```
....
```

Lo stesso accade per “La superba”: ho integrato i dati delle due stelle manualmente.

4 Salviamo STAR_DB per gli usi futuri:

```
In [26]: %store STAR_DB
Stored 'STAR_DB' (list)
```

- L'ultimo atto del passo precedente è consistito nel salvare il risultato ottenuto fin'ora (STAR_DB) usando la direttiva `%store` di `ipython`.
- La lista di oggetti può facilmente essere recuperata con `%store -r` nell'ambiente `ipython`.
- Ma come salvare gli stessi dati in un formato più generale (ad es.: utilizzabile fuori dall'ambiente `ipython`)?
- **JSON**, (JavaScript Object Notation), è un formato nato per l'interscambio di dati fra applicazioni client-server
- Per la sua semplicità viene sempre più spesso usato per lo scambio di dati fra applicazioni generiche
- Il modulo `json` di `python` fornisce funzioni per la scrittura e lettura di file in formato JSON

Generazione del file JSON da STAR_DB:

```
In [1]: %store -r STAR_DB
```

```
In [2]: fpt = open("star_db.json", "w")
```

```
In [3]: import json
```

```
In [4]: json.dump(STAR_DB, fpt)
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-4-b33b583b7fc2> in <module>()  
----> 1 json.dump(STAR_DB, fpt)  
      ...
```

```
TypeError: masked is not JSON serializable
```

Vediamo cosa è successo:

```
In [5]: STAR_DB[0]
```

```
Out[5]:
```

```
['Acamar',  
 'Theta 1 Eridani',  
 '* tet01 Eri',  
 u'02 58 15.696',  
 u'-40 18 16.97',  
 masked,  
 3.54,  
 3.2]
```

```
In [6]: [type(x) for x in STAR_DB[0]]
```

```
Out[6]:
```

```
[str,  
 str,  
 str,  
 numpy.unicode_,  
 numpy.unicode_,  
 numpy.ma.core.MaskedConstant,  
 numpy.float32,  
 numpy.float32]
```

file: convert.py

```
# conversione dati di STAR_DB in formato compatibile JSON

import numpy

def convert(stars):
    for star in stars:
        for ix, f in enumerate(star):
            if type(star[ix]) is numpy.float32:
                star[ix] = float(star[ix])
            elif type(star[ix]) is numpy.ma.core.MaskedConstant:
                star[ix] = 100.
            else:
                star[ix] = unicode(star[ix])
```

Conversione e salvataggio in formato JSON

```
In [2]: import convert
```

```
In [3]: convert.convert(STAR_DB)
```

```
In [4]: STAR_DB[0]
```

```
Out[4]:
```

```
[u'Acamar',
 u'Theta 1 Eridani',
 u'* tet01 Eri',
 u'02 58 15.696',
 u'-40 18 16.97',
 100.0,
 3.5399999618530273,
 3.200000047683716]
```

```
In [5]: import json
```

```
In [6]: fpt = open("star_db.json", "w")
```

```
In [7]: json.dump(STAR_DB, fpt, indent=2)
```

```
In [8]: fpt.close()
```

Nota: Per un approccio più generale vedere: **json.JSONEncoder**

Struttura del file JSON:

```
[
  [
    "Acamar",
    "Theta 1 Eridani",
    "* tet01 Eri",
    "02 58 15.696",
    "-40 18 16.97",
    100.0,
    3.5399999618530273,
    3.200000047683716
  ],
  [
    "Achernar",
    "Alpha Eridani",
    "* alf Eri",
    "01 37 42.8454",
    "-57 14 12.310",
    -0.36000001430511475,
    0.30000001192092896,
    0.46000000834465027
  ],
  ...
]
```

Lettura dati da file JSON:

```
In [1]: import json
```

```
In [2]: fpt = open("star_db.json")
```

```
In [3]: star_db=json.load(fpt)
```

```
In [4]: star_db[22]
```

```
Out[4]:
```

```
[u'Aldhibah',
 u'Zeta Draconis',
 u'* zet Dra',
 u'17 08 47.1959',
 u'+65 42 52.863',
 2.630000114440918,
 3.059999942779541,
 3.1700000762939453]
```

Potrebbe essere utile avere la lista di stelle in ordine di magnitudine:

```
In [5]: star_db.sort(key=lambda x: np.min(x[5:8]))
```

```
In [6]: star_db[0]
```

```
Out[6]:
```

```
[u'Sirius',  
 u'Alpha Canis Majoris',  
 u'* alf CMa',  
 u'06 45 08.9172',  
 u'-16 42 58.017',  
 -1.5099999904632568,  
 -1.4600000381469727,  
 -1.4600000381469727]
```

(Non dimentichiamo di salvare la nuova versione di `star_db`)

file: `altaz.py`

```
import numpy as np
```

```
import astropy.units as u
```

```
from astropy.time import Time
```

```
from astropy.coordinates import SkyCoord, EarthLocation, AltAz
```

```
def atmidnight(star, location, utc_offset):
```

```
    "Calcola altezza dell'oggetto alla mezzanotte per tutto l'anno"
```

```
    h_offset = (utc_offset)*u.hour
```

```
    t0=Time("2017-01-01 00:00:00")-h_offset
```

```
    alldays = t0+np.arange(365)*u.day
```

```
    azsteps = AltAz(obstime=alldays, location=location)
```

```
    radec = " ".join(star[3:5])
```

```
    coords = SkyCoord(radec, unit=(u.hourangle, u.deg))
```

```
    altaz = coords.transform_to(azsteps)
```

```
    return altaz
```

```
firenze = EarthLocation.from_geodetic(lat=43.75*u.deg,
```

```
                                     lon=11.25*u.deg,
```

```
                                     height=40)
```

Vogliamo ottenere un grafico della visibilità di alcune stelle date, ad esempio:

```
In [87]: star_db[0][0]
Out[87]: u'Sirius'
```

```
In [88]: star_db[60][0]
Out[88]: u'Polaris'
```

Esercizio: come trovare l'indice in `star_db` della Polare?

Usiamo la funzione `atmidnight()`:

```
In [80]: %run altaz.py
```

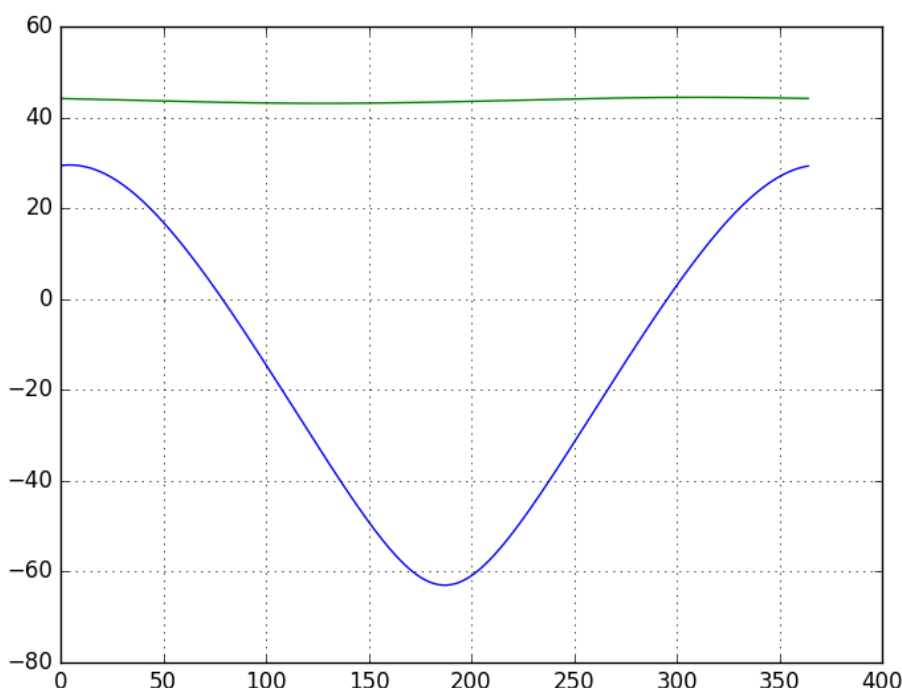
```
In [81]: sirio=atmidnight(star_db[0],firenze,1)
```

```
In [82]: polare=atmidnight(star_db[60],firenze,1)
```

```
In [83]: plt.plot(sirio.alt)
Out[83]: [<matplotlib.lines.Line2D at 0x7f51d79c19d0>]
```

```
In [84]: plt.plot(polaris.alt)
Out[84]: [<matplotlib.lines.Line2D at 0x7f51d762d690>]
```

```
In [85]: plt.grid()
```



Vogliamo rendere l'asse X un po' più utile.

file: `plotlabels.py`

```
import time

def mticks(yy):
    tepoch = [time.mktime((yy,x,1,0,0,0,0,-1)) for x in range(1,13)]
    mdays = [time.localtime(x)[7] for x in tepoch]
    mnames = ["1 gen","1 feb","1 mar","1 apr","1 mag","1 giu",
              "1 lug","1 ago","1 set","1 ott","1 nov","1 dic"]
    return mdays, mnames
```

...

```
In [88]: %run plotlabels.py
```

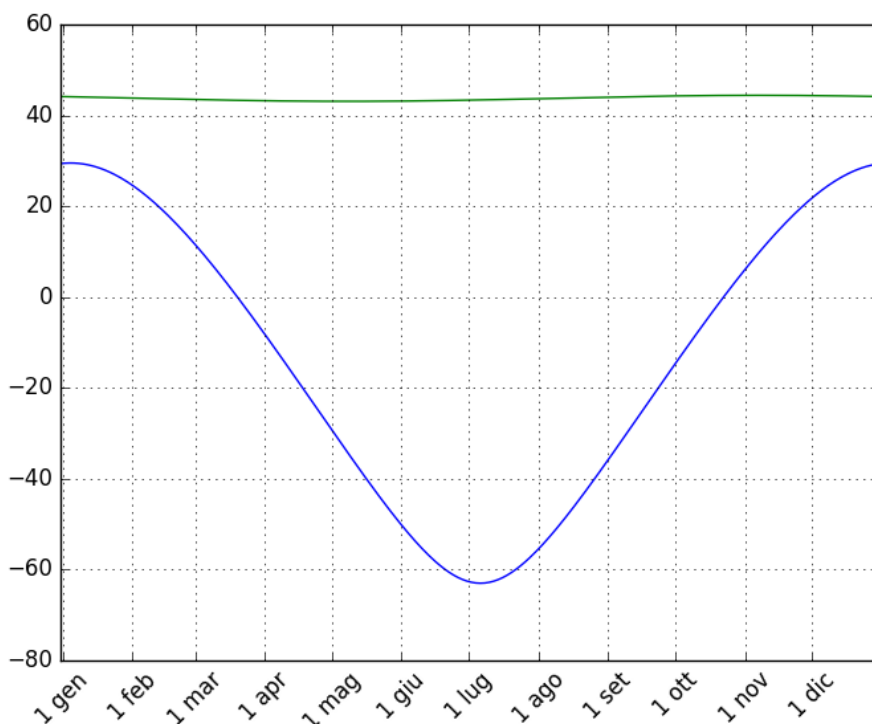
```
In [89]: mdays,mnames = mticks(2017)
```

```
In [90]: ticks = plt.xticks(mdays,mnames)
```

```
In [91]: axes=plt.gca()
```

```
In [92]: plt.xticks(rotation=45)
```

```
In [93]: axes.set_xlim(0,366)
```



file: allplots.py

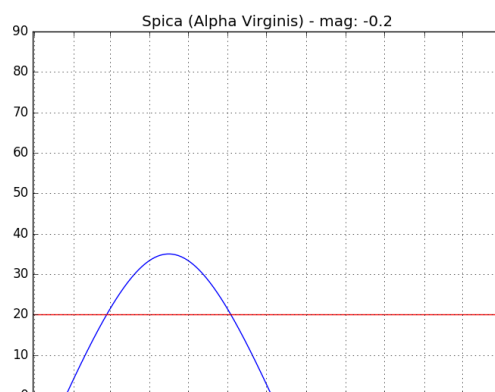
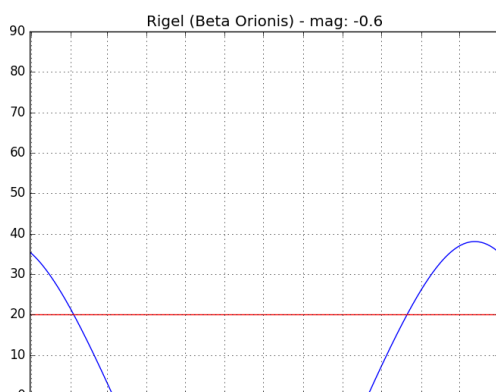
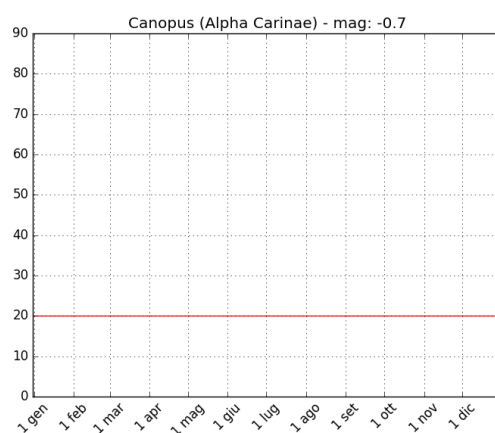
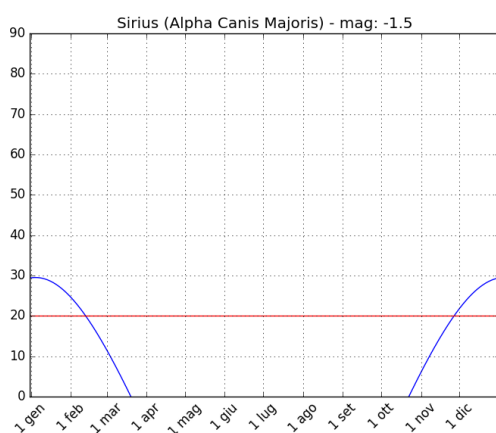
```
import json
import matplotlib.pyplot as plt

from altaz import atmidnight, firenze
from plotlabels import mticks

with open("star_db.json") as fpt:
    stelle = json.load(fpt)

mdays, mnames = mticks(2017)

for s in stelle:
    sdata = atmidnight(s, firenze, 1)
    plt.plot(sdata.alt)
    ticks = plt.xticks(mdays, mnames, rotation=45)
    axes=plt.gca()
    axes.set_xlim(0,366)
    axes.set_ylim(0,90)
    plt.plot((0,366),(20,20),color="r")
    plt.grid()
    mag = min(s[5:8])
    plt.title("%s (%s) - mag: %.1f"%(s[0],s[1],mag))
    plt.show()
```



Procedura per l'invio di e-mail, usabile per:

- Inviare un messaggio ad una lista di indirizzi
- Inviare messaggi di avviso automaticamente (sistemi di segnalazione di fenomeni astronomici estemporanei)
- Inviare messaggi di errore da procedure automatiche
- ...

file: `simplemail.py`

```
#!/usr/bin/python
"""
Il piu' semplice cliente e-mail
"""

import smtplib

def send(mailhost,sender,recipients,subj,body):
    message="""From: %s
To: %s
Subject: %s

""" % (sender,', '.join(recipients), subj) + body

    s = smtplib.SMTP(mailhost)
    s.sendmail(sender, recipients, message)
    s.quit()

if __name__ == '__main__':
    send("smtp.arcetri.astro.it","president@whitehouse.gov",
        ("lfini@arcetri.astro.it"), "Messaggio di prova",
        "Sopra la panca la capra campa")
```

file: webserver.py

```
from __future__ import print_function
from BaseHTTPServer import HTTPServer, BaseHTTPRequestHandler

HEAD = """
<h3>Stage in Astronomia 2017</h3>
<h2>Web server di esempio</h2>
"""

FORM = """
<form action=uso_form>
Scrivi qui <input type=text name=testo>
e premi <input type=submit value=Invia>
</form>
"""

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        try:
            testo=self.path.split("?")[1].split("=")[1]
        except:
            testo=""
        self.send_response(200)
        self.send_header("Content-Type", "text/html")
        self.end_headers()
        print("<!DOCTYPE HTML PUBLIC \"/>\"",
        print("<html><head>", self.wfile)
        print("</head><body>", self.wfile)
        print(HEAD, file=self.wfile)
        if testo:
            print("Hai scritto: %s <p>"%testo, file=self.wfile)
        print(FORM, file=self.wfile)

class WebServer(HTTPServer):
    def __init__(self):
        server_address = ('', 8888)
        HTTPServer.__init__(self, server_address, MyHTTPRequestHandler)

    def serve_forever(self):
        while True:
            self.handle_request()

server = WebServer()
server.serve_forever()
```

Vari package per implementazione di GUI:

- Tkinter/Tix
- PyQt
- kiwy
- wxPython
- ...

file: `gui1.py`

```
import Tix

class MyWidget(Tix.Frame):
    def __init__(self, root):
        Tix.Frame.__init__(self, root)
        self.text=Tix.Text(self)
        self.text.pack(side=Tix.TOP)
        bottom=Tix.Frame(self)
        bottom.pack(side=Tix.TOP)
        b1=Tix.Button(bottom, text="Cancella", command=self.cancella)
        b1.pack(side=Tix.LEFT)
        b2=Tix.Button(bottom, text="Esci", command=root.destroy)
        b2.pack(side=Tix.LEFT)

    def cancella(self):
        self.text.delete(1.0, Tix.END)

root=Tix.Tk()
root.title("Esempio Tix N.1")
wdg=MyWidget(root)
wdg.pack()
root.mainloop()
```

GUI: Programmazione “event driven”

file: gui2.py

```
import sys
import Tix
from threading import Thread

class Input(Thread):
    def __init__(self, wdg):
        Thread.__init__(self)
        self._wdg=wdg
        self.daemon=True

    def run(self):
        while True:
            l=sys.stdin.readline()
            self._wdg.text.insert(Tix.END, l)

class MyWidget(Tix.Frame):
    def __init__(self, root):
        Tix.Frame.__init__(self, root)
        self.text=Tix.Text(self)
        self.text.pack(side=Tix.TOP)
        bottom=Tix.Frame(self)
        bottom.pack(side=Tix.TOP)
        b1=Tix.Button(bottom, text="Cancella", command=self.cancella)
        b1.pack(side=Tix.LEFT)
        b2=Tix.Button(bottom, text="Esci", command=root.destroy)
        b2.pack(side=Tix.LEFT)
    def cancella(self):
        self.text.delete(1.0, Tix.END)

root=Tix.Tk()
root.title("Esempio Tix N.2")
wdg=MyWidget(root)
wdg.pack()
inp=Input(wdg)
inp.start()
print "\nOra scrivi qualche linea ...\n"
root.mainloop()
```

Il grande assente

Una cosa importante che non è stata trattata nel corso di questo seminario:

jupyter

Un sistema che integra l'ambiente ipython con un generatore di codice HTML progettato per la costruzione di pagine interattive e dinamiche in cui coesistono testo, formule, immagini e porzioni di codice scritto in python, o in altri linguaggi.

Bibliografia

- 1 **The Official Python Tutorial.**
<https://docs.python.org/2.7/tutorial>
<https://docs.python.org/3/tutorial/>
- 2 **Python for Data Analysis.** Wes McKinney, O'Reilly.
Tratta gli strumenti avanzati di Python per applicazioni di analisi dati, inclusi: IPython, NumPy, Matplotlib
- 3 **Python Scripting for Computational Science,** H. P. Langtangen, University of Oslo.
Contiene una introduzione più ampia a Python come linguaggio in generale e tratta anche di strumenti avanzati, ma (almeno la versione che ho trovato io) è un po' obsoleta.
- 4 **Guide to NumPy,** Travis E. Oliphant.
Una guida di NumPy abbastanza completa. La versione del 2006 è disponibile liberamente in PDF; la seconda edizione (2015) esiste solo a stampa.
- 5 **Python Crash Course,** Eric Matthes, No Starch Press.
Una introduzione completa al linguaggio, con la descrizione di alcuni progetti, anche complessi di carattere generale. Include un capitolo sull'analisi e la visualizzazione di dati.
- 6 **Dive Into Python,** Mark Pilgrim, on-line.
Un classico manuale avanzato (ma fermo al 2004)
- 7 **Dive Into Python 3,** Mark Pilgrim, on-line.
Come sopra ma relativo a Python 3.
- 8 **Python. Guida completa,** Marco Buttu, Digital Life Style.
Una guida completa al linguaggio con un'ottimo trattamento degli aspetti formali.

- <https://www.python.org/>
- <https://pypi.python.org/pypi>
- <https://www.scipy.org/>
- <http://www.numpy.org/>
- <http://www.astropy.org/>