Well known stars - 1

- For an outreach project I must set up a list of “well known” stars, i.e.: stars with an assigned name, visible with naked eye, belonging to known constellations.

Well known stars - 1

- For an outreach project I must set up a list of “well known” stars, i.e.: stars with an assigned name, visible with naked eye, belonging to known constellations.
- After selecting the main ones, I’d like to know the period of good visibility of each.

Well known stars - 1

- For an outreach project I must set up a list of “well known” stars, i.e.: stars with an assigned name, visible with naked eye, belonging to known constellations.
- After selecting the main ones, I’d like to know the period of good visibility of each.

How did I proceed:

Well known stars - 1

- For an outreach project I must set up a list of “well known” stars, i.e.: stars with an assigned name, visible with naked eye, belonging to known constellations.
- After selecting the main ones, I’d like to know the period of good visibility of each.

How did I proceed:

1. From some web site[1], and with hand editing I’ve written down a list of names into file `starlist.py`.

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

Well known stars - 1

- For an outreach project I must set up a list of “well known” stars, i.e.: stars with an assigned name, visible with naked eye, belonging to known constellations.
- After selecting the main ones, I’d like to know the period of good visibility of each.

How did I proceed:

1. From some web site[1], and with hand editing I’ve written down a list of names into file `starlist.py`.

file: `starlist.py`

```
# Star list with name and canonical name

#        Traditional name      Long name
STAR_DB = [["Acamar",          "Theta 1 Eridani"],
         ["Achernar",          "Alpha Eridani"],
         ["Achird",            "Eta Cassiopeiae"],
....
         ["Zubenelgenubi",     "Alpha 2 Librae"],
         ["Zubeneschamali",    "Beta Librae"]]
```

[1] 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

Well known stars - 1

- For an outreach project I must set up a list of “well known” stars, i.e.: stars with an assigned name, visible with naked eye, belonging to known constellations.
- After selecting the main ones, I’d like to know the period of good visibility of each.

How did I proceed:

1. From some web site[1], and with hand editing I’ve written down a list of names into file `starlist.py`.

[1] 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

file: `starlist.py`

```
# Star list with name and canonical name

#        Traditional name       Long name
STAR_DB = [["Acamar",           "Theta 1 Erid
           ["Achernar",         "Alpha Eridan
           ["Achird",           "Eta Cassiopeiae"],
....
           ["Zubenelgenubi",    "Alpha 2 Librae"],
           ["Zubeneschamali",   "Beta Librae"]]
```

For simplicity the file contains valid python code, i.e.: star names are written as a python list

Well known stars - 2

2. For the next steps I've used the `Simbad` module from `astroquery` in order to:
   - Verify the existence of the name
   - Retrieve star parameters (including the "canonical" name)

Well known stars - 2

2. For the next steps I’ve used the Simbad module from astroquery in order to:
   - Verify the existence of the name
   - Retrieve star parameters (including the “canonical” name)

file: starinfo.py

```
import sys, time
from astroquery.simbad import Simbad
from starlist import STAR_DB

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

for star in STAR_DB:
    time.sleep(1)
    try:
        obj = Simbad.query_object(star[0])
    except Exception as e:
        print("Error:", 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): not found" % (star[0], star[1]))
```

Well known stars - 2

2. For the next steps I’ve used the `Simbad` module from `astroquery` in order to:
   - Verify the existence of the name
   - Retrieve star parameters (including the “canonical” name)

file: `starinfo.py`

Star fluxes are not included by default in the Simbad response, so I request them

```
import sys, time
from astroquery.simbad import Simbad
from starlist import STAR_DB

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

for star in STAR_DB:
    time.sleep(1)
    try:
        obj = Simbad.query_object(star[0])
    except Exception as e:
        print("Error:", 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): not found" % (star[0], star[1]))
```

Well known stars - 2

2. For the next steps I’ve used the `Simbad` module from `astroquery` in order to:
   - Verify the existence of the name
   - Retrieve star parameters (including the “canonical” name)

file: `starinfo.py`

```
import sys, time
from astroquery.simbad import Simbad
from starlist import STAR_DB

Simbad.add_votable_fie

for star in STAR_DB:
    time.sleep(1)
    try:
        obj = Simbad.query_object(star[0])
    except Exception as e:
        print("Error:", 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): not found" % (star[0], star[1]))
```

Star fluxes are not included by default in the Simbad response, so I request them

The delay is needed because the Simbad server rejects request sequences which are too fast (to avoid DoS attacks)

Well known stars - 2

2. For the next steps I’ve used the `Simbad` module from `astroquery` in order to:
    - Verify the existence of the name
    - Retrieve star parameters (including the “canonical” name)

file: `starinfo.py`

Star fluxes are not included by default in the Simbad response, so I request them

```
import sys, time
from astroquery.simbad import Simbad
from starlist import STAR_DB

Simbad.add_votable_fie

for star in STAR_DB:
    time.sleep(1)
    try:
        obj = Simbad.query_object(star[0])
    except Exception as e:
        print("Error:", 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_da
            print(star)
        else:
            print( "%s (%s): not found" % (star[0], star[1]))
```

The delay is needed because the Simbad server rejects request sequences which are too fast (to avoid DoS attacks)

Print out star data

Well known stars - 3

Executing starinfo.py

```
In [1]: %run starinfo.py
['Acamar', 'Theta 1 Eridani', b'* tet01 Eri', '02 58 15.715', '-40 18 17.03', ma
['Achernar', 'Alpha Eridani', b'* alf Eri', '01 37 42.8454', '-57 14 12.310', -0
['Achird', 'Eta Cassiopeiae', b'* eta Cas', '00 49 06.2907', '+57 48 54.675', 4.
 ....
Al Dhanab (Gamma Gruis): not found
 ....
['Zubenelgenubi', 'Alpha 2 Librae', b'* alf02 Lib', '14 50 52.7130', '-16 02 30.
['Zubeneschamali', 'Beta Librae', b'* bet Lib', '15 17 00.4138', '-09 22 58.491'
```

Well known stars - 3

Executing starinfo.py

```
In [1]: %run starinfo.py
['Acamar', 'Theta 1 Eridani', b'* tet01 Eri', '02 58 15.715', '-40 18 17.03', ma
['Achernar', 'Alpha Eridani', b'* alf Eri', '01 37 42.8454', '-57 14 12.310', -0
['Achird', 'Eta Cassiopeiae', b'* eta Cas', '00 49 06.2907', '+57 48 54.675', 4.
 ....
Al Dhanab (Gamma Gruis): not found
 ....
['Zubenelgenubi', 'Alpha 2                     ', '14 50 52.7130', '-16 02 30.
['Zubeneschamali', 'Beta Librae', b'* bet Lib', '15 17 00.4138', '-09 22 58.491'
```

⇑

What’s that **b**?
... see next slide

Well known stars - 3

Executing starinfo.py

```
In [1]: %run starinfo.py
['Acamar', 'Theta 1 Eridani', b'* tet01 Eri', '02 58 15.715', '-40 18 17.03', ma
['Achernar', 'Alpha Eridani', b'* alf Eri', '01 37 42.8454', '-57 14 12.310', -0
['Achird', 'Eta Cassiopeiae', b'* eta Cas', '00 49 06.2907', '+57 48 54.675', 4.
 ....
Al Dhanab (Gamma Gruis): not found
 ....
['Zubenelgenubi', 'Alpha                        ', '14 50 52.7130', '-16 02 30.
['Zubeneschamali', 'Beta Librae', b'* bet Lib', '15 17 00.4138', '-09 22 58.491'
```

⇑ What's that **b**?
... see next slide

Notes:

- For all names not found (e.g.: Al Dhanab) I made some test in order to verify whether:
  - The name is not listed in Simbad
  - The name is listed with different spelling
  - Some other error

  Then I edited file `starlist.py` accordingly.

Well known stars - 3

Executing `starinfo.py`

```
In [1]: %run starinfo.py
['Acamar', 'Theta 1 Eridani', b'* tet01 Eri', '02 58 15.715', '-40 18 17.03', ma
['Achernar', 'Alpha Eridani', b'* alf Eri', '01 37 42.8454', '-57 14 12.310', -0
['Achird', 'Eta Cassiopeiae', b'* eta Cas', '00 49 06.2907', '+57 48 54.675', 4.
 ....
Al Dhanab (Gamma Gruis): not found
 ....
['Zubenelgenubi', 'Alpha                         ', '14 50 52.7130', '-16 02 30.
['Zubeneschamali', 'Beta Librae', b'* bet Lib', '15 17 00.4138', '-09 22 58.491'
```

⇑

What's that **b**?
... see next slide

Notes:

- For all names not found (e.g.: Al Dhanab) I made some test in order to verify whether:
  - The name is not listed in Simbad
  - The name is listed with different spelling
  - Some other error

  Then I edited file `starlist.py` accordingly.
- From Simbad database I got the “canonical name”, the coordinates and the fluxes.

Detour 1: strings in python3 /1

- Python 3 provides a formal support for strings in all the languages.

Detour 1: strings in python3 /1

- Python 3 provides a formal support for strings in all the languages.
  - All strings are **unicode** strings

Detour 1: strings in python3 /1

- Python 3 provides a formal support for strings in all the languages.
  - All strings are **unicode** strings
  - A new type has been defined: **bytes**

Detour 1: strings in python3 /1

- Python 3 provides a formal support for strings in all the languages.
  - All strings are **unicode** strings
  - A new type has been defined: **bytes**
  - Input/output of strings must specify the *encoding*, to convert unicode into bytes and vice-versa

Detour 1: strings in python3 /1

- Python 3 provides a formal support for strings in all the languages.
  - All strings are **unicode** strings
  - A new type has been defined: **bytes**
  - Input/output of strings must specify the *encoding*, to convert unicode into bytes and vice-versa
  - The most used encoding for western languages is UTF-8.

Detour 1: strings in python3 /1

- Python 3 provides a formal support for strings in all the languages.
  - All strings are **unicode** strings
  - A new type has been defined: **bytes**
  - Input/output of strings must specify the *encoding*, to convert unicode into bytes and vice-versa
  - The most used encoding for western languages is UTF-8.
- In python 2 unicode support is via the `unicode` module
  - The type *str* was used both for strings and for byte arrays

Detour 1: strings in python3 /1

- Python 3 provides a formal support for strings in all the languages.
  - All strings are **unicode** strings
  - A new type has been defined: **bytes**
  - Input/output of strings must specify the *encoding*, to convert unicode into bytes and vice-versa
  - The most used encoding for western languages is UTF-8.
- In python 2 unicode support is via the `unicode` module
  - The type *str* was used both for strings and for byte arrays

Examples

```
$ python
>>> a1 = "abcd"
>>> a2 = b"abcd"
>>> type(a1)
<class 'str'>
>>> type(a2)
<class 'bytes'>
>>> a2[0]+1
98
>>> a1[0]+1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Detour 1: strings in python3 /1

- Python 3 provides a formal support for strings in all the languages.
  - All strings are **unicode** strings
  - A new type has been defined: **bytes**
  - Input/output of strings must specify the *encoding*, to convert unicode into bytes and vice-versa
  - The most used encoding for western languages is UTF-8.
- In python 2 unicode support is via the `unicode` module
  - The type *str* was used both for strings and for byte arrays

Examples

```
$ python
>>> a1 = "abcd"
>>> a2 = b"abcd"
>>> type(a1)
<class ’str’>
>>> type(a2)
<class ’bytes’>
>>> a2[0]+1
98
>>> a1[0]+1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can’t convert ’int’ object to str implicitly
```

a1 is a plain string

Detour 1: strings in python3 /1

- Python 3 provides a formal support for strings in all the languages.
  - All strings are **unicode** strings
  - A new type has been defined: **bytes**
  - Input/output of strings must specify the *encoding*, to convert unicode into bytes and vice-versa
  - The most used encoding for western languages is UTF-8.
- In python 2 unicode support is via the `unicode` module
  - The type *str* was used both for strings and for byte arrays

Examples

```
$ python
>>> a1 = "abcd"
>>> a2 = b"abcd"
>>> type(a1)
<class 'str'>
>>> type(a2)
<class 'bytes'>
>>> a2[0]+1
98
>>> a1[0]+1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

a1 is a plain string

a2 is a byte string (values are ASCII codes of characters)

Detour 1: strings in python3 /1

- Python 3 provides a formal support for strings in all the languages.
  - All strings are **unicode** strings
  - A new type has been defined: **bytes**
  - Input/output of strings must specify the *encoding*, to convert unicode into bytes and vice-versa
  - The most used encoding for western languages is UTF-8.
- In python 2 unicode support is via the `unicode` module
  - The type *str* was used both for strings and for byte arrays

Examples

```
$ python
>>> a1 = "abcd"
>>> a2 = b"abcd"
>>> type(a1)
<class 'str'>
>>> type(a2)
<class 'bytes'>
>>> a2[0]+1
98
>>> a1[0]+1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

a1 is a plain string

a2 is a byte string (values are ASCII codes of characters)

a2 elements are integers in [0-255]

Detour 1: strings in python3 /1

- Python 3 provides a formal support for strings in all the languages.
  - All strings are **unicode** strings
  - A new type has been defined: **bytes**
  - Input/output of strings must specify the *encoding*, to convert unicode into bytes and vice-versa
  - The most used encoding for western languages is UTF-8.
- In python 2 unicode support is via the `unicode` module
  - The type *str* was used both for strings and for byte arrays

Examples

```
$ python
>>> a1 = "abcd"
>>> a2 = b"abcd"
>>> type(a1)
<class 'str'>
>>> type(a2)
<class 'bytes'>
>>> a2[0]+1
98
>>> a1[0]+1
Traceback (most recent ca
  File "<stdin>", line 1,
TypeError: Can't convert 'int' object to str implicitly
```

a1 is a plain string

a2 is a byte string (values are ASCII codes of characters)

a2 elements are integers in [0-255]

a1 elements are characters

Detour 1: strings in python3 /2

Interpolation/formatting of strings

Detour 1: strings in python3 /2

Interpolation/formatting of strings

- The traditional % operator:

```
a = "String with number: %d and text: %s" % (11, "eleven")
```

Detour 1: strings in python3 /2

Interpolation/formatting of strings

- The traditional **%** operator:

```
a = "String with number: %d and text: %s" % (11, "eleven")
```

- The **format** method:

```
b = "String with number: {} and text: {}".format(11, "eleven")
c = "String with number: {nn} and text: {tt}".format(nn=11, tt="eleven")
```

Detour 1: strings in python3 /2

Interpolation/formatting of strings

- The traditional **%** operator:

```
a = "String with number: %d and text: %s" % (11, "eleven")
```

- The **format** method:

```
b = "String with number: {} and text: {}".format(11, "eleven")
c = "String with number: {nn} and text: {tt}".format(nn=11, tt="eleven")
```

- The “**f**” special string (python 3.6):

```
nn = 11
tt = "eleven"
d = f"String with number: {nn} and text: {tt}"
```

Detour 1: strings in python3 /2

Interpolation/formatting of strings

- The traditional **%** operator:

```
a = "String with number: %d and text: %s" % (11, "eleven")
```

- The **format** method:

```
b = "String with number: {} and text: {}".format(11, "eleven")
c = "String with number: {nn} and text: {tt}".format(nn=11, tt="eleven")
```

- The “**f**” special string (python 3.6):

```
nn = 11
tt = "eleven"
d = f"String with number: {nn} and text: {tt}"
```

Both the `format()` method and the f-string, allow more complex specification of how to format the interpolated values.

Detour 1: strings in python3 /2

Interpolation/formatting of strings

- The traditional **%** operator:

```
a = "String with number: %d and text: %s" % (11, "eleven")
```

- The **format** method:

```
b = "String with number: {} and text: {}".format(11, "eleven")
c = "String with number: {nn} and text: {tt}".format(nn=11, tt="eleven")
```

- The “**f**” special string (python 3.6):

```
nn = 11
tt = "eleven"
d = f"String with number: {nn} and text: {tt}"
```

Both the `format()` method and the f-string, allow more complex specification of how to format the interpolated values.

→

Well known stars - 4

3 A funny error in Simbad:

```
In [2]: aldanab = Simbad.query_object("al dhanab")
/usr/local/lib/python3.6/dist-packages/astroquery/simbad/core.py:136: UserWarnin
  (error.line, error.msg))

In [3]: aldanab = Simbad.query_object("gamma gruis")

In [4]: aldanab
Out[4]:
<Table masked=True length=1>
 MAIN_ID      RA          DEC     RA_PREC DEC_PREC COO_ERR_MAJA COO_ERR_MINA ...
           "h:m:s"     "d:m:s"                          mas         mas     ...
  object     str13       str13     int16  int16    float32     float32    ...
--------- ------------- ------------- ------- -------- ------------ ------------
* gam Gru 21 53 55.7262 -37 21 53.479 9       9       5.280      3.520 ...

In [5]: Simbad.query_objectids("gam gru")
Out[5]:
<Table length=32>
         ID
       bytes23
-----------------------
        NAME Al Dhanab
             PLX 5287
            * gam Gru
....
```

Well known stars - 4

3 A funny error in Simbad:

```
In [2]: aldanab = Simbad.query_object("al dhanab")
/usr/local/lib/python3.6/dist-packages/astroque                        erWarnin
  (error.line, error.msg))

In [3]: aldanab = Simbad.query_object("gamma gruis")

In [4]: aldanab
Out[4]:
<Table masked=True length=1>
 MAIN_ID      RA          DEC      RA_PREC DEC_PREC COO_ERR_MAJA COO_ERR_MINA ...
            "h:m:s"     "d:m:s"                         mas          mas      ...
  object     str13       str13      int16   int16     float32      float32    ...
--------- ------------- ------------- -------- --------- ------------ ------------
* gam Gru 21 53 55.7262 -37 21 53.479 9       9        5.280        3.520 ...

In [5]: Simbad.query_objectids("gam gru")
Out[5]:
<Table length=32>
         ID
       bytes23
-----------------------
        NAME Al Dhanab
             PLX 5287
            * gam Gru
....
```

The search for "al dhanab" fails

Well known stars - 4

3 A funny error in Simbad:

```
In [2]: aldanab = Simbad.query_object("al dhanab")
/usr/local/lib/python3.6/dist-packages/astroque                     erWarnin
  (error.line, error.msg))

In [3]: aldanab = Simbad.query_object("gamma gruis")

In [4]: aldanab
Out[4]:
<Table masked=True length=1>
 MAIN_ID       RA            DEC       RA_PREC DEC_PREC COO_ERR_MAJA COO_ERR_MINA ...
            "h:m:s"       "d:m:s"                           mas          mas      ...
  object     str13         str13      int16   int16      float32      float32    ...
--------- ------------- ------------- -------- --------- ------------ ------------
* gam Gru 21 53 55.7262 -37 21 53.479 9       9        5.280       3.520 ...

In [5]: Simbad.query_objectids("gam gru")
Out[5]:
<Table length=32>
         ID
       bytes23
-----------------------
        NAME Al Dhanab
             PLX 5287
            * gam Gru
....
```

The search for "al dhanab" fails

But the star exists, e.g.: as "gamma gruis"

Well known stars - 4

3 A funny error in Simbad:

```
In [2]: aldanab = Simbad.query_object("al dhanab")
/usr/local/lib/python3.6/dist-packages/astroque                    erWarnin
  (error.line, error.msg))

In [3]: aldanab = Simbad.query_object("gamma gruis")

In [4]: aldanab
Out[4]:
<Table masked=True length=1>
 MAIN_ID      RA           DEC      RA_PREC DEC_PREC COO_ERR_MAJA COO_ERR_MINA ...
           "h:m:s"      "d:m:s"                          mas          mas      ...
  object     str13        str13      int16   int16     float32      float32    ...
--------- ------------- ------------- -------- --------- ------------ ------------
* gam Gru 21 53 55.7262 -37 21 53.479 9       9        5.280       3.520 ...

In [5]: Simbad.query_objectids("gam gru")
Out[5]:
<Table length=32>
          ID
       bytes23
-----------------------
        NAME Al Dhanab
             PLX 5287
            * gam Gru
....
```

The search for "al dhanab" fails

But the star exists, e.g.: as "gamma gruis"

And "al dhanab" is listed as alternative name

Well known stars - 4

3 A funny error in Simbad:

```
In [2]: aldanab = Simbad.query_object("al dhanab")
/usr/local/lib/python3.6/dist-packages/astroque                     erWarnin
  (error.line, error.msg))

In [3]: aldanab = Simbad.query_object("gamma gruis")

In [4]: aldanab
Out[4]:
<Table masked=True length=1>
 MAIN_ID      RA          DEC      RA_PREC DEC_PREC COO_ERR_MAJA COO_ERR_MINA ...
           "h:m:s"      "d:m:s"                         mas          mas      ...
  object     str13        str13      int16   int16     float32      float32    ...
--------- ------------- ------------- -------- --------- ------------ ------------
* gam Gru 21 53 55.7262 -37 21 53.479 9       9        5.280       3.520 ...

In [5]: Simbad.query_objectids("gam gru")
Out[5]:
<Table length=32>
         ID
       bytes23
-----------------------
       NAME Al Dhanab
            PLX 5287
           * gam Gru
....
```

The search for "al dhanab" fails

But the star exists, e.g.: as "gamma gruis"

And "al dhanab" is listed as alternative name

For this and a few similar cases I've edited proper values by hand

Detour 2: JSON - 1

JSON format - 7

4. Now we save the star list (variable STAR_DB) for future use:

```
In [6]: %store STAR_DB
Stored ’STAR_DB’ (list)
```

Detour 2: JSON - 1

4. Now we save the star list (variable `STAR_DB`) for future use:

```
In [6]: %store STAR_DB
Stored ’STAR_DB’ (list)
```

- `STAR_DB` can be easily restored into the `ipython` environment with:
  `%store -r`.

Detour 2: JSON - 1

4. Now we save the star list (variable `STAR_DB`) for future use:

```
In [6]: %store STAR_DB
Stored 'STAR_DB' (list)
```

- `STAR_DB` can be easily restored into the `ipython` environment with:

  `%store -r`.
- Sometimes anyway it might be better to store a variable in a more general format (e.g.: to use it from languages other than python)

Detour 2: JSON - 1

4. Now we save the star list (variable `STAR_DB`) for future use:

```
In [6]: %store STAR_DB
Stored ’STAR_DB’ (list)
```

- `STAR_DB` can be easily restored into the `ipython` environment with:

  `%store -r`.
- Sometimes anyway it might be better to store a variable in a more general format (e.g.: to use it from languages other than python)
- **JSON** (**J**ava**S**cript **O**bject **N**otation) is a format used for data exchange in client-server applications.

Detour 2: JSON - 1

4. Now we save the star list (variable `STAR_DB`) for future use:

```
In [6]: %store STAR_DB
Stored 'STAR_DB' (list)
```

- `STAR_DB` can be easily restored into the `ipython` environment with:

  `%store -r`.
- Sometimes anyway it might be better to store a variable in a more general format (e.g.: to use it from languages other than python)
- **JSON** (**J**ava**S**cript **O**bject **N**otation) is a format used for data exchange in client-server applications.
- Thanks to its simplicity is is often used for data exchange in many other contexts.

Detour 2: JSON - 1

4. Now we save the star list (variable `STAR_DB`) for future use:

```
In [6]: %store STAR_DB
Stored 'STAR_DB' (list)
```

- `STAR_DB` can be easily restored into the `ipython` environment with:

  `%store -r`.
- Sometimes anyway it might be better to store a variable in a more general format (e.g.: to use it from languages other than python)
- **JSON** (**J**ava**S**cript **O**bject **N**otation) is a format used for data exchange in client-server applications.
- Thanks to its simplicity is is often used for data exchange in many other contexts.
- Python support for JSON data read/write is provided by module: `json`.

Generating a JSON file from STAR_DB:

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

In [2]: import json

In [3]: with open("star_db.json","w") as fpt:
   ...:     json.dump(STAR_DB,fpt)
   ...:
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-8b34f18b3695> in <module>()
      1 with open("star_db.json","w") as fpt:
----> 2     json.dump(STAR_DB,fpt)
 ....
TypeError: b'* tet01 Eri' is not JSON serializable

In [4]:
```

Detour 2: JSON - 2

JSON format - 8

Generating a JSON file from STAR_DB:

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

In [2]: import json

In [3]: with open("star_db.json","w") as fpt:
   ...:     json.dump(STAR_DB,fpt)
   ...:
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-8b34f18b3695> in <module>()
      1 with open("star_db.json","w") as fpt:
----> 2     json.dump(STAR_DB,fpt)
 ....
TypeError: b'* tet01 Eri' is not JSON serializable

In [4]:
```

Problem: not every python object is “JSON serializable”

Detour 2: JSON - 2

Generating a JSON file from STAR_DB:

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

In [2]: import json

In [3]: with open("star_db.json","w") as fpt:
   ...:   json.dump(STAR_DB,fpt)
   ...:
---------------------------------------------------------------------------
TypeError                           Traceback (most recent call last)
<ipython-input-3-8b34f18b3695> in <module>()
     1 with open("star_db.json","w") as fpt:
----> 2   json.dump(STAR_DB,fpt)
 ....
TypeError: b'* tet01 Eri' is not JSON serializable

In [4]:
```

Problem: not every python object is “JSON serializable”

With further analysis we would find more non JSON serializable types into STAR_DB.

Detour 2: JSON - 2

Generating a JSON file from STAR_DB:

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

In [2]: import json

In [3]: with open("star_db.json","w") as fpt:
   ...:     json.dump(STAR_DB,fpt)
   ...:
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-8b34f18b3695> in <module>()
      1 with open("star_db.json","w") as fpt:
----> 2     json.dump(STAR_DB,fpt)
 ....
TypeError: b'* tet01 Eri' is not JSON serializable

In [4]:
```

Problem: not every python object is “JSON serializable”

With further analysis we would find more non JSON serializable types into STAR_DB.

```
In [4]: STAR_DB[0]
Out[4]:
['Acamar',
 'Theta 1 Eridani',
 b'* tet01 Eri',
 '02 58 15.715',
 '-40 18 17.03',
 masked,
 3.3299999,
 3.1800001]

In [5]: type(STAR_DB[0][5])
Out[5]: numpy.ma.core.MaskedConstant

In [6]: type(STAR_DB[0][6])
Out[6]: numpy.float32
```

Detour 2: JSON - 2

Generating a JSON file from STAR_DB:

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

In [2]: import json

In [3]: with open("star_db.json","w") as fpt:
   ...:     json.dump(STAR_DB,fpt)
   ...:
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-8b34f18b3695> in <module>()
      1 with open("star_db.json","w") as fpt:
----> 2     json.dump(STAR_DB,fpt)
 ....
TypeError: b’* tet01 Eri’ is not JSON serializable

In [4]:
```

Problem: not every python object is “JSON serializable”

With further analysis we would find more non JSON serializable types into STAR_DB.

```
In [4]: STAR_DB[0]
Out[4]:
[’Acamar’,
 ’Theta 1 Eridani’,
 b’* tet01 Eri’,
 ’02 58 15.715’,
 ’-40 18 17.03’,
 masked,
 3.3299999,
 3.1800001]

In [5]: type(STAR_DB[0][5])
Out[5]: numpy.ma.core.MaskedConstant

In [6]: type(STAR_DB[0][6])
Out[6]: numpy.float32
```

The **Simbad** module uses the numpy defined type MaskedConstant for values not defined, and numpy.float32 for float numbers

Detour 2: JSON - 3

file: convert.py

```python
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.
            elif type(star[ix]) is bytes:
                star[ix] = star[ix].decode("ascii")
```

Detour 2: JSON - 3

file: convert.py

```
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.
            elif type(star[ix]) is bytes:
                star[ix] = star[ix].decode("ascii")
```

Convert `numpy.float32` into *float*

Detour 2: JSON - 3

file: convert.py

```
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
            elif type(star[ix]) is nu
                star[ix] = 100.
            elif type(star[ix]) is bytes:
                star[ix] = star[ix].decode("ascii")
```

Convert numpy.float32

Convert MaskedConstant into an “impossible” value

Detour 2: JSON - 3

file: convert.py

```
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[
            elif type(star[ix]) is nu
                star[ix] = 100.
            elif type(star[ix]) is bytes:
                star[ix] = star[ix].decode("ascii")
```

Convert numpy.float32

Convert MaskedConstant
into an “impossible” value

Convert bytes
into a string

Detour 2: JSON - 3

file: convert.py

```
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[
            elif type(star[ix]) is nu
                star[ix] = 100.
            elif type(star[ix]) is bytes:
                star[ix] = star[ix].decode("ascii")
```

Convert numpy.float32

Convert MaskedConstant into an “impossible” value

Convert bytes into a string

Converting STAR_DB to be JSON serializable:

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

In [2]: %run convert.py

In [3]: convert(STAR_DB)

In [4]: STAR_DB[0]
Out[4]:
['Acamar',
 'Theta 1 Eridani',
 '* tet01 Eri',
  ...
In [5]: type(STAR_DB[0][5])
Out[5]: float

In [6]: type(STAR_DB[0][6])
Out[6]: float
```

Detour 2: JSON - 3

file: convert.py

```
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[
            elif type(star[ix]) is nu
                star[ix] = 100.
            elif type(star[ix]) is bytes:
                star[ix] = star[ix].decode("ascii")
```

Convert numpy.float32

Convert MaskedConstant into an “impossible” value

Convert bytes into a string

Converting STAR_DB to be JSON serializable:

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

In [2]: %run convert.py

In [3]: convert(STAR_DB)

In [4]: STAR_DB[0]
Out[4]:
['Acamar',
 'Theta 1 Eridani',
 '* tet01 Eri',
  ...
In [5]: type(STAR_DB[0][5])
Out[5]: float

In [6]: type(STAR_DB[0][6])
Out[6]: float
```

Detour 2: JSON - 3

file: convert.py

```
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[
            elif type(star[ix]) is nu
                star[ix] = 100.  ⟸
            elif type(star[ix]) is bytes:
                star[ix] = star[ix].decode("ascii")  ⟸
```

Convert numpy.float32

Convert MaskedConstant
into an “impossible” value

Convert bytes
into a string

Converting STAR_DB to be JSON serializable:

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

In [2]: %run convert.py

In [3]: convert(STAR_DB)

In [4]: STAR_DB[0]
Out[4]:
['Acamar',
 'Theta 1 Eridani',
 '* tet01 Eri',  ⟸
  ...
In [5]: type(STAR_DB[0][5])
Out[5]: float  ⟸

In [6]: type(STAR_DB[0][6])
Out[6]: float
```

Detour 2: JSON - 3

file: convert.py

```
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[
            elif type(star[ix]) is nu
                star[ix] = 100.
            elif type(star[ix]) is bytes:
                star[ix] = star[ix].decode("ascii")
```

Convert numpy.float32

Convert MaskedConstant into an “impossible” value

Convert bytes into a string

Converting STAR_DB to be JSON serializable:

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

In [2]: %run convert.py

In [3]: convert(STAR_DB)

In [4]: STAR_DB[0]
Out[4]:
['Acamar',
 'Theta 1 Eridani',
 '* tet01 Eri',
  ...
In [5]: type(STAR_DB[0][5])
Out[5]: float

In [6]: type(STAR_DB[0][6])
Out[6]: float
```

Detour 2: JSON - 3

JSON format - 9

file: convert.py

```
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[
            elif type(star[ix]) is nu
                star[ix] = 100.
            elif type(star[ix]) is bytes:
                star[ix] = star[ix].decode("ascii")
```

Convert numpy.float32

Convert MaskedConstant into an “impossible” value

Convert bytes into a string

Converting STAR_DB to be JSON serializable:

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

In [2]: %run convert.py

In [3]: convert(STAR_DB)

In [4]: STAR_DB[0]
Out[4]:
['Acamar',
 'Theta 1 Eridani',
 '* tet01 Eri',
  ...
In [5]: type(STAR_DB[0][5])
Out[5]: float

In [6]: type(STAR_DB[0][6])
Out[6]: float
```

Note: For a more general approach see: `json.JSONEncoder`

Detour 2: JSON - 4

After conversion STAR_DB can be written as JSON:

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

In [2]: %run convert.py

In [3]: convert(STAR_DB)

In [4]: import json

In [5]: with open("star_db.json","w") as fpt:
   ...:     json.dump(STAR_DB,fpt,indent=2)
   ...:
```

Detour 2: JSON - 4

After conversion STAR_DB can be written as JSON:

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

In [2]: %run convert.py

In [3]: convert(STAR_DB)

In [4]: import json

In [5]: with open("star_db.json","w") as fpt:
   ...:    json.dump(STAR_DB,fpt,indent=2)
   ...:
```

The `indent` value results in a more readable format of the JSON file

Detour 2: JSON - 4

After conversion STAR_DB can be written as JSON:

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

In [2]: %run convert.py

In [3]: convert(STAR_DB)

In [4]: import json

In [5]: with open("star_db.json","w") as fpt:
   ...:    json.dump(STAR_DB,fpt,indent=2)
   ...:
```

The indent value results in a more readable format of the JSON file

What does a JSON file look like:

```
In [6]: !more star_db.json
[
  [
    "Acamar",
    "Theta 1 Eridani",
    "* tet01 Eri",
    "02 58 15.715",
    "-40 18 17.03",
    100.0,
    3.3299999237060547,
    3.180000066757202
  ],
  [
    "Achernar",
    "Alpha Eridani",
    "* alf Eri",
 ....
```

Detour 2: JSON - 4

JSON format - 10

After conversion STAR_DB can be written as JSON:

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

In [2]: %run convert.py

In [3]: convert(STAR_DB)

In [4]: import json

In [5]: with open("star_db.json","w") as fpt:
   ...:    json.dump(STAR_DB,fpt,indent=2)
   ...:
```

The indent value results in a more readable format of the JSON file

What does a JSON file look like:

```
In [6]: !more star_db.json
[
  [
    "Acamar",
    "Theta 1 Eridani",
    "* tet01 Eri",
    "02 58 15.715",
    "-40 18 17.03",
    100.0,
    3.3299999237060547,
    3.180000066757202
  ],
  [
    "Achernar",
    "Alpha Eridani",
    "* alf Eri",
 ....
```

The JSON standard can represent hierarchical structures like python *lists* and *dictionaries*.

Scalar values are either decimal numbers or strings

Well known stars - 5

Let’s go on working with the JSON file:

```
In [1]: import json

In [2]: with open("star_db.json") as fpt:
   ...:    star_db=json.load(fpt)
   ...:

In [3]: star_db.sort(key=lambda x: np.min(x[5:8]))

In [4]: star_db[0]
Out[4]:
['Sirius',
 'Alpha Canis Majoris',
 '* alf CMa',
 '06 45 08.9172',
 '-16 42 58.017',
 -1.5099999904632568,
 -1.4600000381469727,
 -1.4600000381469727]
```

Well known stars - 5

Let’s go on working with the JSON file:

```
In [1]: import json

In [2]: with open("star_db.json") as fp
   ...:     star_db=json.load(fpt)
   ...:

In [3]: star_db.sort(key=lambda x: np.min(x[5:8]))

In [4]: star_db[0]
Out[4]:
['Sirius',
 'Alpha Canis Majoris',
 '* alf CMa',
 '06 45 08.9172',
 '-16 42 58.017',
 -1.5099999904632568,
 -1.4600000381469727,
 -1.4600000381469727]
```

⟸ Here’s how you read back data from a JSON file

Well known stars - 5

Let’s go on working with the JSON file:

```
In [1]: import json

In [2]: with open("star_db.json") as fp
   ...:     star_db=json.load(fpt)
   ...:

In [3]: star_db.sort(key=lambda x: np.min(x[5:8]))

In [4]: star_db[0]
Out[4]:
['Sirius',
 'Alpha Canis Majoris',
 '* alf CMa',
 '06 45 08.9172',
 '-16 42 58.017',
 -1.5099999904632568,
 -1.4600000381469727,
 -1.4600000381469727]
```

Here’s how you read back data from a JSON file

Let’s sort stars in order of magnitude

Well known stars - 5

Let’s go on working with the JSON file:

```
In [1]: import json

In [2]: with open("star_db.json") as fp
   ...:    star_db=json.load(fpt)
   ...:

In [3]: star_db.sort(key=lambda x: np.min(x[5:8]))

In [4]: star_db[0]
Out[4]:
['Sirius',
 'Alpha Canis Majoris',
 '* alf CMa',
 '06 45 08.9172',
 '-16 42 58.017',
 -1.5099999904632568,
 -1.4600000381469727,
 -1.4600000381469727]
```

Here’s how you read back data from a JSON file

Let’s sort stars in order of magnitude

As you might expect the topmost star is Sirius

Well known stars - 5

Let’s go on working with the JSON file:

```
In [1]: import json

In [2]: with open("star_db.json") as fp
   ...:     star_db=json.load(fpt)
   ...:

In [3]: star_db.sort(key=lambda x: np.min(x[5:8]))

In [4]: star_db[0]
Out[4]:
['Sirius',
 'Alpha Canis Majoris',
 '* alf CMa',
 '06 45 08.9172',
 '-16 42 58.017',
 -1.5099999904632568,
 -1.4600000381469727,
 -1.4600000381469727]
```

Here’s how you read back data from a JSON file

Let’s sort stars in order of magnitude

As you might expect the topmost star is Sirius

Let’s proceed with the next step:

file: altaz.py

```python
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):
    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)
```

Well known stars - 5

Let’s go on working with the JSON file:

```
In [1]: import json

In [2]: with open("star_db.json") as fp
   ...:     star_db=json.load(fpt)
   ...:

In [3]: star_db.sort(key=lambda x: np.min(x[5:8]))

In [4]: star_db[0]
Out[4]:
['Sirius',
 'Alpha Canis Majoris',
 '* alf CMa',
 '06 45 08.9172',
 '-16 42 58.017',
 -1.5099999904632568,
 -1.4600000381469727,
 -1.4600000381469727]
```

Here’s how you read back data from a JSON file

Let’s sort stars in order of magnitude

As you might expect the topmost star is Sirius

Let’s proceed with the next step:

file: altaz.py

```
import numpy as np
import astropy.units as u
from astropy.time import Time
from astropy.coordinates import SkyCoord,

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

function `atmidnight()` computes a star alt-azimut coordinates at local midnight for every day of a year, from a given location on earth

Well known stars - 5

Let’s go on working with the JSON file:

```
In [1]: import json

In [2]: with open("star_db.json") as fp
   ...:     star_db=json.load(fpt)
   ...:

In [3]: star_db.sort(key=lambda x: np.min(x[5:8]))

In [4]: star_db[0]
Out[4]:
['Sirius',
 'Alpha Canis Majoris',
 '* alf CMa',
 '06 45 08.9172',
 '-16 42 58.017',
 -1.5099999904632568,
 -1.4600000381469727,
 -1.4600000381469727]
```

Here’s how you read back data from a JSON file

Let’s sort stars in order of magnitude

As you might expect the topmost star is Sirius

Let’s proceed with the next step:

file: altaz.py

```
import numpy as np
import astropy.units as u
from astropy.time import Time
from astropy.coordinates import SkyCoord,

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

function `atmidnight()` computes a star alt-azimut coordinates at local midnight for every day of a year, from a given location on earth

Here we also set up the proper location object

Well known stars - 6

My purpose is to draw a plot of visibility of some stars, E.g.: Sirius and Polaris.

Well known stars - 6

My purpose is to draw a plot of visibility of some stars, E.g.: Sirius and Polaris.

Let’s use function `atmidnight()`:

```
In [1]: import json

In [2]: with open("star_db.json") as fpt:
   ...:     star_db=json.load(fpt)
   ...:

In [3]: star_db.sort(key=lambda x: np.min(x[5:8]))

In [4]: %run altaz.py

In [5]: sirius=atmidnight(star_db[0],firenze,1)

In [6]: polaris=atmidnight(star_db[59],firenze,1)

In [7]: plt.plot(sirius.alt)

In [8]: plt.plot(polaris.alt)

In [9]: plt.grid()
```

Well known stars - 6

My purpose is to draw a plot of visibility of some stars, E.g.: Sirius and Polaris.

Let’s use function `atmidnight()`:

```
In [1]: import json

In [2]: with open("star_db.json") as fpt:
   ...:     star_db=json.load(fpt)
   ...:

In [3]: star_db.sort(key=lambda x: np.min(x[5:8]))

In [4]: %run altaz.py

In [5]: sirius=atmidnight(star_db[0],firenze,1)

In [6]: polaris=atmidnight(star_db[59],firenze,1)

In [7]

In [8]

In [9]: plt.grid()
```

Q: How do I get (easily) the index of Polaris into list `star_db`?

A: `[x[0] for x in star_db].index("Polaris")`

Well known stars - 6

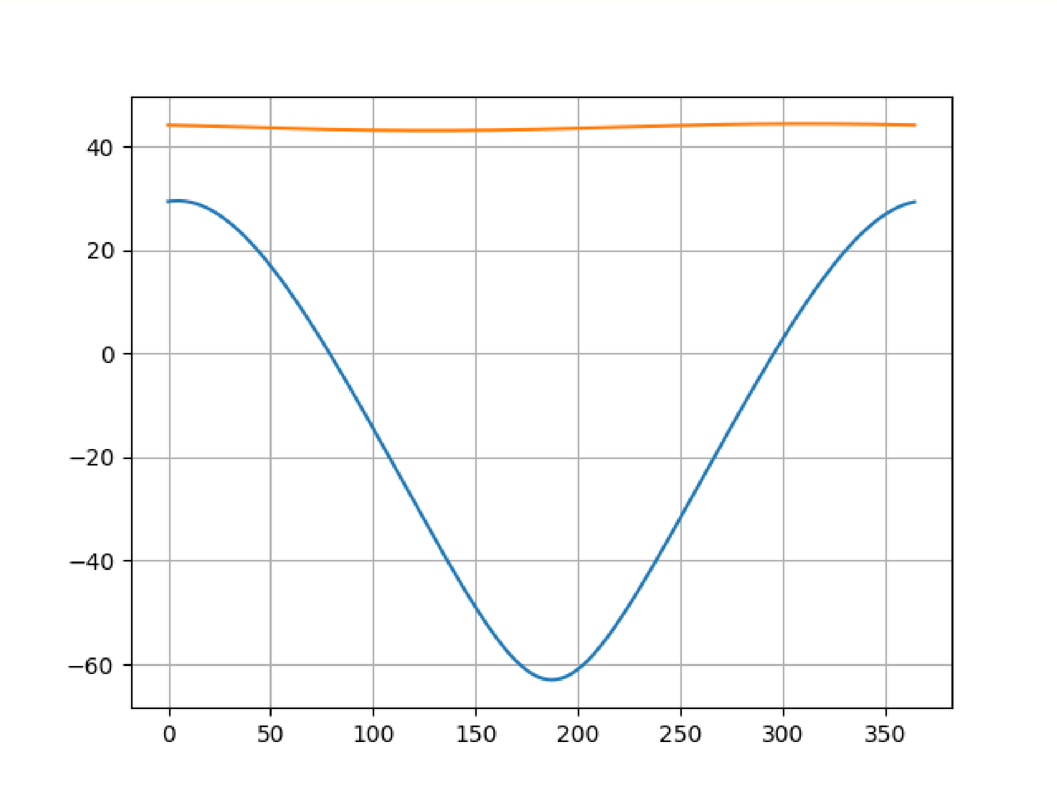
My purpose is to draw a plot of visibility of some stars, E.g.: Sirius and Polaris.

Let's use function `atmidnight()`:

```
In [1]: import json

In [2]: with open("star_db.json") as fpt:
   ...:    star_db=json.load(fpt)
   ...:

In [3]: star_db.sort(key=lambda x: np.min(x[5:8]))

In [4]: %run altaz.py

In [5]: sirius=atmidnight(star_db[0],firenze,1)

In [6]: polaris=atmidnight(star_db[59],firenze,1)

In [7]

In [8]

In [9]: plt.grid()
```

Q: How do I get (easily) the index of Polaris into list `star_db`?

A: `[x[0] for x in star_db].index("Polaris")`



Well known stars - 7

Doing the plot a little better ...

Well known stars - 7

Doing the plot a little better ...

file: `plotlabels.py`

```
import time

def mticks(yy):
    tepoch = [time.mktime((yy,x,1,0,0,0,0,0,-1)) for x in range(1,13)]
    mdays = [time.localtime(x)[7] for x in tepoch]
    mnames = ["Jan 1","Feb 1","Mar 1","Apr 1","May 1","Jun 1",
              "Jul 1","Aug 1","Sep 1","Oct 1","Nov 1","Dec 1"]
    return mdays, mnames
```

Doing the plot a little better ...

file: plotlabels.py

```
import time

def mticks(yy):
    tepoch = [time.mktime((yy,x,1,0,0,0,0,0,-1)) for x in range(1,13)]
    mdays = [time.localtime(x)[7] for x in tepoch]
    mnames = ["Jan 1","Feb 1","Mar 1","Apr 1","May 1","Jun 1",
              "Jul 1","Aug 1","Sep 1","Oct 1","Nov 1","Dec 1"]
    return mdays, mnames
```

...

```
In [10]: %run plotlabels.py

In [11]: mdays,mnames = mticks(2018)

In [12]: ticks = plt.xticks(mdays,mnames)

In [13]: axes=plt.gca()

In [14]: plt.xticks(rotation=45)

In [15]: axes.set_xlim(0,366)
```

Doing the plot a little better ...

file: `plotlabels.py`

```
import time

def mticks(yy):
    tepoch = [time.mktime((yy,x,1,0,0,0,0,0,-1)) for x in range(1,13)]
    mdays = [time.localtime(x)[7] for x in tepoch]
    mnames = ["Jan 1","Feb 1","Mar 1","Apr 1","May 1","Jun 1",
             "Jul 1","Aug 1","Sep 1","Oct 1","Nov 1","Dec 1"]
    return mdays, mnames
```

...

```
In [10]: %run plotlabels.py

In [11]: mdays,mnames = mticks(2018)

In [12]: ticks = plt.xticks(mdays,mnames)

In [13

In [14

In [15]: axes.set_xlim(0,366)
```

mdays: `[1, 32, 60, ...]`

mnames: `["Jan 1", "Feb 1", "Mar 1", ...]`

Well known stars - 7

Doing the plot a little better ...

file: `plotlabels.py`

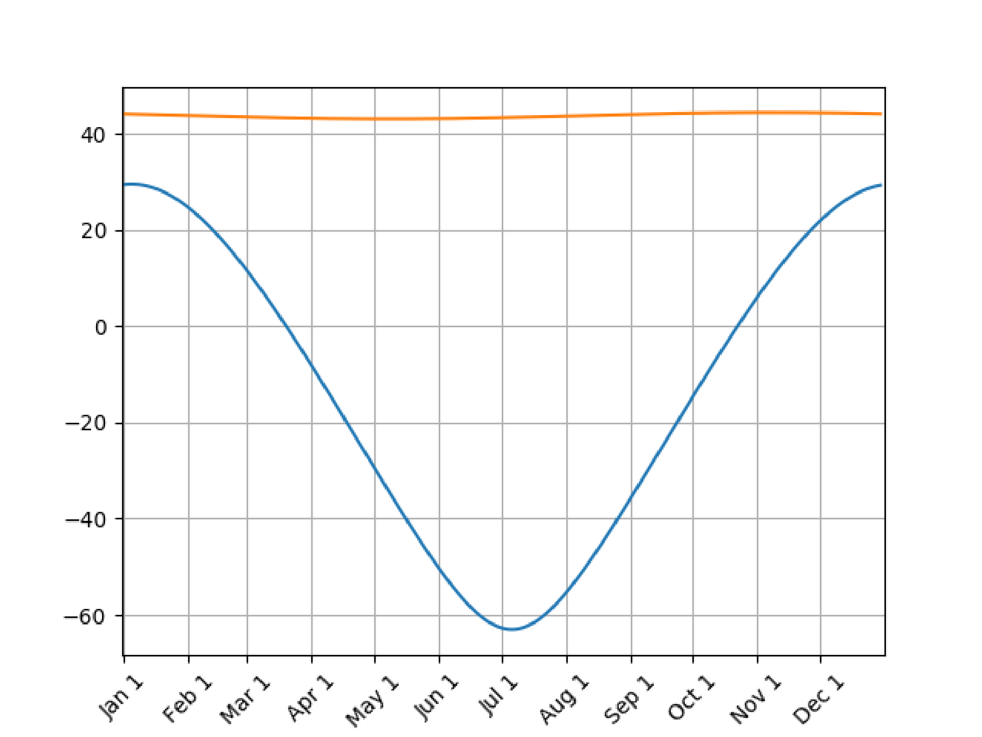
```python
import time

def mticks(yy):
    tepoch = [time.mktime((yy,x,1,0,0,0,0,0,-1)) for x in range(1,13)]
    mdays = [time.localtime(x)[7] for x in tepoch]
    mnames = ["Jan 1","Feb 1","Mar 1","Apr 1","May 1","Jun 1",
              "Jul 1","Aug 1","Sep 1","Oct 1","Nov 1","Dec 1"]
    return mdays, mnames
```

...

```
In [10]: %run plotlabels.py

In [11]: mdays,mnames = mticks(2018)

In [12]: ticks = plt.xticks(mdays,mnames)

In [13

In [14

In [15]: axes.set_xlim(0,366)
```

mdays: `[1, 32, 60, ...]`

mnames: `["Jan 1", "Feb 1", "Mar 1", ...]`



Miscellaneous

A collection of miscellaneous examples

Sending e-mail

Useful for:

- Sending messages to mailing lists
- Sending automated announces
- Sending error messages from procedures
- ...

Sending e-mail

Useful for:

- Sending messages to mailing lists
- Sending automated announces
- Sending error messages from procedures
- ...

file: simplemail.py

```
#!/usr/bin/python
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",), "Test message",
         "The quick brown fox jumps over the lazy dog")
```

Sending e-mail

Useful for:

- Sending messages to mailing lists
- Sending automated announces
- Sending error messages from procedures
- ...

file: `simplemail.py`

```python
#!/usr/bin/python
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",), "Test message",
         "The quick brown fox jumps over the lazy dog")
```

Note: for this example we've used a server which does not require authentication.

The `smtplib` module, anyway, supports also TLS authentication

Simple Web Server

file: webserver.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler

HEAD = """<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html><head>
</head><body>
<h3>Python for Astronomy 2018</h3>
<h2>Simple Web Server</h2>""".encode("utf8")
FORM = """<form action=uso_form>
Write here <input type=text name=text>
and press <input type=submit value=Send>
</form>""".encode("utf8")

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        try:
            text=self.path.split("?")[1].split("=")[1]
        except:
            text=""
        self.send_response(200)
        self.send_header(b"Content-Type", "text/html")
        self.end_headers()
        self.wfile.write(HEAD)
        if text:
           msg = "You wrote: %s <p>"%text
           self.wfile.write(msg.encode("utf8"))
        self.wfile.write(FORM)

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()
```

Simple Web Server

file: webserver.py

```python
from http.server import HTTPServer, BaseHTTPRequestHandler

HEAD = """<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html><head>
</head><body>
<h3>Python for Astronomy 2018</h3>
<h2>Simple Web Server</h2>""".encode("utf8")
FORM = """<form action=uso_form>
Write here <input type=text name=text>
and press <input type=submit value=Send>
</form>""".encode("utf8")

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        try:
            text=self.path.split("?")[1].split("=")[1]
        except:
            text=""
        self.send_response(200)
        self.send_header(b"Content-Type", "text/html")
        self.end_headers()
        self.wfile.write(HEAD)
        if text:
           msg = "You wrote: %s <p>"%text
           self.wfile.write(msg.encode("utf8"))
        self.wfile.write(FORM)

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()
```

We implement the server deriving from class HTTPServer and implementing method: serve_forever()

Simple Web Server

file: webserver.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler

HEAD = """<!DOCTYPE HTML PUB                                    /EN">
<html><head>
</head><body>
<h3>Python for Astronomy 2018</h3>
<h2>Simple Web Server</h2>""".encode("utf8")
FORM = """<form action=uso_form>
Write here <input type=text name=text>
and press <input type=submit value=Send>
</form>""".encode("utf8")

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        try:
            text=self.path.split("?")[1].split("=")[1]
        except:
            text=""
        self.send_response(200)
        self.send_header(b"Content-Type", "text/html")
        self.end_headers()
        self.wfile.write(HEAD)
        if text:
           msg = "You wrote: %s <p>"%text
           self.wfile.write(msg.encode("utf8"))
        self.wfile.write(FORM)

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()
```

HEAD: the top, fixed part of web page

We implement the server deriving from class HTTPServer and implementing method: serve_forever()

Simple Web Server

file: webserver.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler

HEAD = """<!DOCTYPE HTML PUB                                        /EN">
<html><head>
</head><body>
<h3>Python for Astronomy 2018</h3>
<h2>Simple Web Server</h2>""".encode("utf8")
FORM = """<form action=uso_form>
Write here <input type=text name=t
and press <input type=submit value
</form>""".encode("utf8")

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        try:
            text=self.path.split("?")[1].split("=")[1]
        except:
            text=""
        self.send_response(200)
        self.send_header(b"Content-Type", "text/html")
        self.end_headers()
        self.wfile.write(HEAD)
        if text:
           msg = "You wrote: %s <p>"%text
           self.wfile.write(msg.encode("utf8"))
        self.wfile.write(FORM)

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()
```

`HEAD`: the top, fixed part of web page

`FORM`: the bottom, fixed part of web page

We implement the server deriving from class `HTTPServer` and implementing method: `serve_forever()`

Simple Web Server

file: webserver.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler

HEAD = """<!DOCTYPE HTML PUB                                         /EN">
<html><head>
</head><body>
<h3>Python for Astronomy 2018</h3>
<h2>Simple Web Server</h2>""".encode("utf8")
FORM = """<form action=uso_form>
Write here <input type=text name=t
and press <input type=submit value
</form>""".encode("utf8")

class MyHTTPRequestHandler(BaseHTT
    def do_GET(self):
        try:
            text=self.path.split("?")[1].split("=")[1]
        except:
            text=""
        self.send_response(200)
        self.send_header(b"Content-Type", "text/html")
        self.end_headers()
        self.wfile.write(HEAD)
        if text:
           msg = "You wrote: %s <p>"%text
           self.wfile.write(msg.encode("utf8"))
        self.wfile.write(FORM)

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()
```

`HEAD`: the top, fixed part of web page

`FORM`: the bottom, fixed part of web page

⟸ Method `do_Get()`: is called to ”serve” each client request

We implement the server deriving from class `HTTPServer` and implementing method: `serve_forever()`

Simple Web Server

file: webserver.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler

HEAD = """<!DOCTYPE HTML PUB                                        /EN">
<html><head>
</head><body>
<h3>Python for Astronomy 2018</h3>
<h2>Simple Web Server</h2>""".encode("utf8")
FORM = """<form action=uso_form>
Write here <input type=text name=t
and press <input type=submit value
</form>""".encode("utf8")

class MyHTTPRequestHandler(BaseHTT
    def do_GET(self):
        try:
            text=self.path.split("?")[1].split("=")[1
        except:
            text=""
        self.send_response(200)
        self.send_header(b"Content-Type", "text/html")
        self.end_headers()
        self.wfile.write(HEAD)
        if text:
           msg = "You wrote: %s <p>"%text
           self.wfile.write(msg.encode("utf8"))
        self.wfile.write(FORM)

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()
```

HEAD: the top, fixed part of web page

FORM: the bottom, fixed part of web page

⇐ Method do_Get(): is called to ”serve” each client request

⇐ Prepare required HTTP header

We implement the server deriving from class HTTPServer and implementing method: serve_forever()

Simple Web Server

file: webserver.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler

HEAD = """<!DOCTYPE HTML PUB                                        /EN">
<html><head>
</head><body>
<h3>Python for Astronomy 2018</h3>
<h2>Simple Web Server</h2>""".encode("utf8")
FORM = """<form action=uso_form>
Write here <input type=text name=t
and press <input type=submit value
</form>""".encode("utf8")

class MyHTTPRequestHandler(BaseHTT
    def do_GET(self):
        try:
            text=self.path.split("?")[1].split("=")[1
        except:
            text=""
        self.send_response(200)
        self.send_header(b"Content-Type", "text/html")
        self.end_headers()
        self.wfile.write(HEAD)
        if text:
           msg = "You wrote: %s <p>"%text
           self.wfile.write(msg.encode("utf8"))
        self.wfile.write(FORM)

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()
```

`HEAD`: the top, fixed part of web page

`FORM`: the bottom, fixed part of web page

Method `do_Get()`: is called to ”serve” each client request

Prepare required HTTP header

Write back HTML code

We implement the server deriving from class `HTTPServer` and implementing method: `serve_forever()`

GUI programming - 1

In python you have several choices for GUI programming:

- Tkinter/Tix
- PyQT
- kiwy
- wxPython
- ...

GUI programming - 1

In python you have several choices for GUI programming:

- Tkinter/Tix
- PyQT
- kiwy
- wxPython
- ...

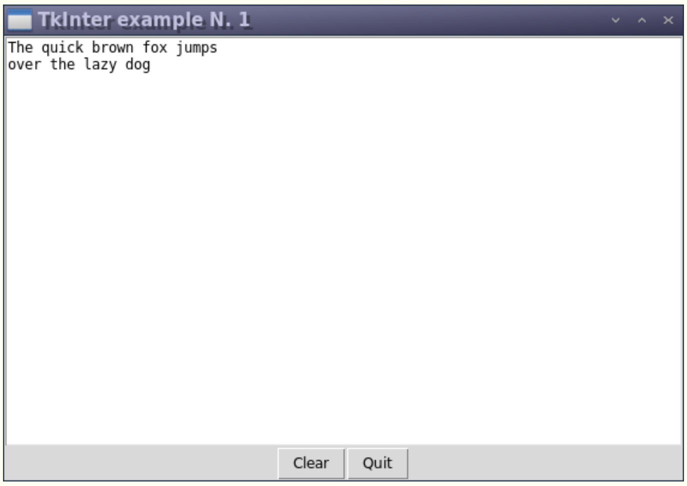
file: gui1.py

```
import tkinter as tk

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

root=tk.Tk()
root.title("TkInter example N. 1")
wdg=MyWidget(root)
wdg.pack()
root.mainloop()
```

GUI programming - 1

In python you have several choices for GUI programming:

- Tkinter/Tix
- PyQT
- kiwy
- wxPython
- ...



TkInter example N. 1

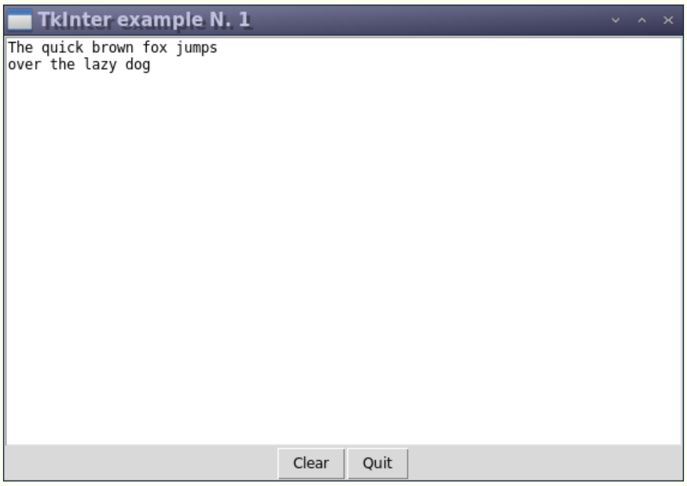
```
The quick brown fox jumps
over the lazy dog
```

file: gui1.py

```python
import tkinter as tk

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

root=tk.Tk()
root.title("TkInter example N. 1")
wdg=MyWidget(root)
wdg.pack()
root.mainloop()
```

In python you have several choices for GUI programming:

- Tkinter/Tix
- PyQT
- kiwy
- wxPython
- ...


TkInter example N. 1

The quick brown fox jumps
over the lazy dog

file: gui1.py

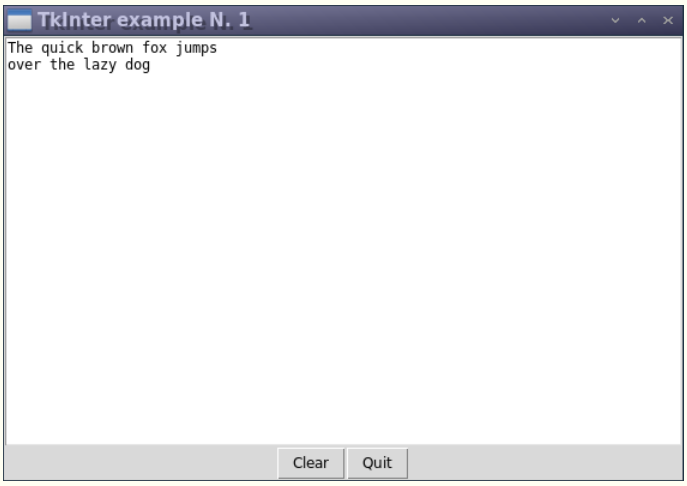
```
import tkinter as tk

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

root=tk.Tk()
root.title("TkInter example N. 1")
wdg=MyWidget(root)
wdg.pack()
root.mainloop()
```

1. Define a sub widget

GUI programming - 1

In python you have several choices for GUI programming:

- Tkinter/Tix
- PyQT
- kiwy
- wxPython
- ...



```
The quick brown fox jumps
over the lazy dog
```

file: gui1.py

```
import tkinter as tk

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

root=tk.Tk()
root.title("TkInter example N. 1")
wdg=MyWidget(root)
wdg.pack()
root.mainloop()
```

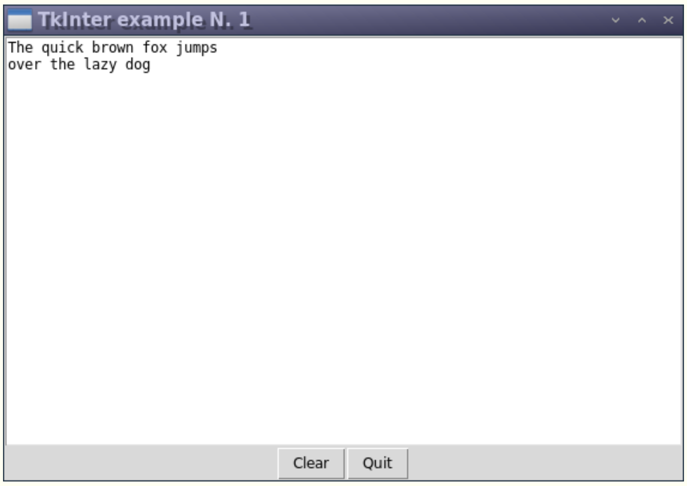
1. Define a sub widget

2. put it in place

GUI programming - 1

In python you have several choices for GUI programming:

- Tkinter/Tix
- PyQT
- kiwy
- wxPython
- ...



```
The quick brown fox jumps
over the lazy dog
```

file: `gui1.py`

```
import tkinter as tk

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

root=tk.Tk()
root.title("TkInter exam
wdg=MyWidget(root)
wdg.pack()
root.mainloop()
```

1. Define a sub widget

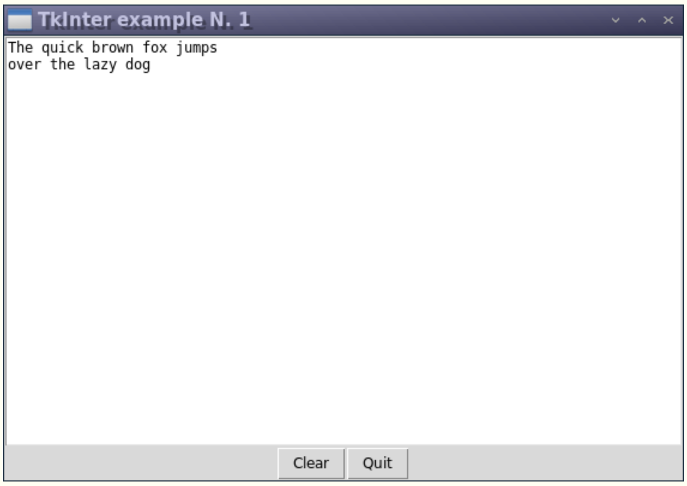
2. put it in place

Instantiate the main widget

GUI programming - 1

In python you have several choices for GUI programming:

- Tkinter/Tix
- PyQT
- kiwy
- wxPython
- ...



The quick brown fox jumps
over the lazy dog

file: gui1.py

```
import tkinter as tk

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

root=tk.Tk()
root.title("TkInter exam
wdg=MyWidget(root)
wdg.pack()
root.mainloop()
```

1. Define a sub widget

2. put it in place

Instantiate the main widget

Start GUI internal loop

GUI programming - 2

GUI programming style is **event driven**

GUI programming style is **event driven**

file: gui2.py

```
import sys
from threading import Thread
import tkinter as tk

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(tk.END,l)

class MyWidget(tk.Frame):
    def __init__(self, root):
        tk.Frame.__init__(self, root)
        self.text=tk.Text(self)
        self.text.pack(side=tk.TOP)
        bottom=tk.Frame(self)
        bottom.pack(side=tk.TOP)
        b1=tk.Button(bottom, text="Clear", command=self.clear)
        b1.pack(side=tk.LEFT)
        b2=tk.Button(bottom, text="Quit", command=root.destroy)
        b2.pack(side=tk.LEFT)

    def clear(self):
        self.text.delete(1.0, tk.END)

root=tk.Tk()
root.title("TkInter example N. 2")
wdg=MyWidget(root)
wdg.pack()
inp=Input(wdg)
inp.start()
print("\nNow write some lines ...\n")
root.mainloop()
```

GUI programming style is **event driven**

file: gui2.py

The threading module supports the *multithreading* programming style.

In our case we use it to have two ”main loops” running concurrently

```
import sys
from threading import Thread
import tkinter as tk

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(tk.END,l)

class MyWidget(tk.Frame):
    def __init__(self, root):
        tk.Frame.__init__(self, root)
        self.text=tk.Text(self)
        self.text.pack(side=tk.TOP)
        bottom=tk.Frame(self)
        bottom.pack(side=tk.TOP)
        b1=tk.Button(bottom, text="Clear", command=self.clear)
        b1.pack(side=tk.LEFT)
        b2=tk.Button(bottom, text="Quit", command=root.destroy)
        b2.pack(side=tk.LEFT)

    def clear(self):
        self.text.delete(1.0, tk.END)

root=tk.Tk()
root.title("TkInter example N. 2")
wdg=MyWidget(root)
wdg.pack()
inp=Input(wdg)
inp.start()
print("\nNow write some lines ...\n")
root.mainloop()
```

GUI programming - 2

More examples - 18

GUI programming style is **event driven**

file: gui2.py

```
import sys
from threading import Thread
import tkinter as tk

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(tk.END,l)

class MyWidget(tk.Frame):
    def __init__(self, root):
        tk.Frame.__init__(self, root)
        self.text=tk.Text(self)
        self.text.pack(side=tk.TOP)
        bottom=tk.Frame(self)
        bottom.pack(side=tk.TOP)
        b1=tk.Button(bottom, text="Clear", command=self.clear)
        b1.pack(side=tk.LEFT)
        b2=tk.Button(bottom, text="Quit", command=root.destroy)
        b2.pack(side=tk.LEFT)

    def clear(self):
        self.text.delete(1.0, tk.END)

root=tk.Tk()
root.title("TkInter example N. 2")
wdg=MyWidget(root)
wdg.pack()
inp=Input(wdg)
inp.start()
print("\nNow write s
root.mainloop()
```

First loop

The `threading` module supports the *multithreading* programming style.

In our case we use it to have two ”main loops” running concurrently

GUI programming style is **event driven**

file: gui2.py

```
import sys
from threading import Thread
import tkinter as tk

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(tk.END,l)

class MyWidget(tk.Frame):
    def __init__(self, root):
        tk.Frame.__init__(self, root)
        self.text=tk.Text(self)
        self.text.pack(side=tk.TOP)
        bottom=tk.Frame(self)
        bottom.pack(side=tk.TOP)
        b1=tk.Button(bottom, text="Clear", command=self.clear)
        b1.pack(side=tk.LEFT)
        b2=tk.Button(bottom, text="Quit", command=root.destroy)
        b2.pack(side=tk.LEFT)

    def clear(self):
        self.text.delete(1.0, tk.END)

root=tk.Tk()
root.title("TkInter example N. 2")
wdg=MyWidget(root)
wdg.pack()
inp=Input(wdg)
inp.start()
print("\nNow write s
root.mainloop()
```

The `threading` module supports the *multithreading* programming style.

In our case we use it to have two ”main loops” running concurrently

Second loop

First loop