F21SC Industrial Programming: Python: Python Libraries

Hans-Wolfgang Loidl

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh

Semester 1 2017/18

[0]No proprietary software has been used in producing these slides

Selected library functions

- One of the main reasons why Python is successful is the rich set of libraries
- This includes standard libraries, that come with a Python distribution, but also third-party libraries
- Prominent third-party libraries are:
  - JSON
  - matplotlib
  - tkinter
  - numpy
  - scipy
  - sympy
  - orange
  - pandas

String libraries and regular expressions

- Python, as many scripting languages, has powerful support for **regular expressions**
- Regular expression can be used to search for strings, replace text etc
- The syntax for regular expression is similar across languages
- For working experience with regular expressions, see this section of the Linux Introduction or these slides on regular expressions.
- There are many good textbooks on regular expressions around.

Basic usage of string libraries and regular expressions

- To **access** the regular expression library use: `import re`
- To **search** for a `substr` in `str` use: `re.search(substr,str)`
- To **replace** a `pattern` by a `repstr` in `string` use: `re.sub(pattern, repstr, string)`
- To **split** a `string`string into `sep`-separated components use: `re.split(pattern,string)`
- Check the Python library documentation for details and more functions.

Examples of regular expressions in Python

Read from a file, print all lines with 'read' event types:

Example

```
file='/home/hwloidl/tmp/sample_10k_lines.json'
print ("Reading from ", file)
with open(file,"r") as f:
    for line in f:
        if (re.search('"event_type":"read"', line)):
            print (line)
```

Pick-up the code from the sample sources section

Examples of regular expressions in Python

Read from a file, split the line, and print one element per line

Example

```
file='/home/hwloidl/tmp/sample_10k_lines.json'
print ("Reading from ", file)
with open(file,"r") as f:
    for line in f:
        if (re.search('"event_type":"read"', line)):
            line0 = re.sub("[{}]", "", line)      # remove {
            for x in re.split("[ ]*,[ ]*",line0):# split by
                print (re.sub(':','->', x))       # replace
```

Saving structured data with JSON

- JSON (JavaScript Object Notation) is a popular, light-weight data exchange format.
- Many languages support this format, thus it's useful for data exchange across systems.
- It is much ligher weight than XML, and thus easier to use.
- `json.dump(x, f)` turns `x` into a string in JSON format and writes it to file `f`.
- `x = json.load(f)` reads `x` from the file `f`, assuming JSON format.
- For detail on the JSON format see: `http://json.org/`

JSON Example

Example

```
tel = dict([('guido', 4127), ('jack', 4098)])
ppTelDict(tel)

# write dictionary to a file in JSON format
json.dump(tel, fp=open(jfile,'w'), indent=2)
print("Data has been written to file ", jfile);

# read file in JSON format and turn it into a dictionary
tel_new = json.loads(open(jfile,'r').read())
ppTelDict(tel_new)

# test a lookup
the_name = "Billy"
printNoOf(the_name,tel_new);
```

Visualisation using `matplotlib`

`matplotlib` is a widely used library for plotting data in various kinds of formats. Advantages of the library are

- It supports a huge range of graphs, such as plots, histograms, power spectra, bar charts, errorcharts, scatterplots etc
- It provides interfaces to external tools such as MATLAB
- It is widely used and well-documented
- For detailed documentation see: Matplotlib documentation

Examples of using `matplotlib`

The following code displays a histogram in horizontal format, with hard-wired data:

Example

```
import matplotlib.pyplot as plt
...
# # horizontal bars: very simple, fixed input
plt.barh([1,2,3], [22,33,77], align='center', alpha=0.4)
#         indices  values
plt.show()
```

Pick-up the code from Sample sources (simple_histo.py)

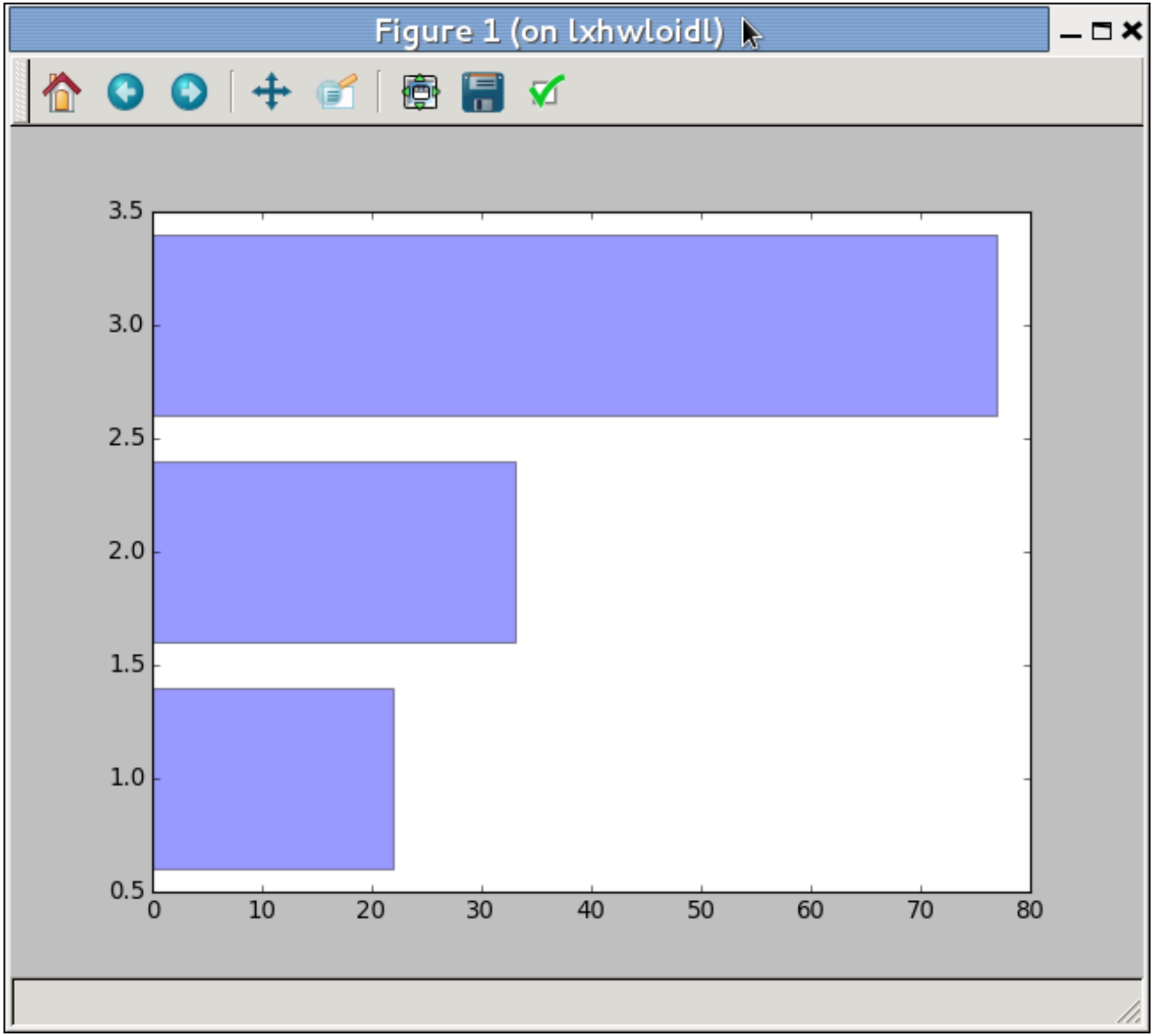
Figure 1 (on lxhwloidl)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
10
20
30
40
50
60
70
80

Examples of using `matplotlib`

A similar examples, with labels:

Example

```
import matplotlib.pyplot as plt
...
# horizontal bars: very simple, fixed input; labels
plt.barh(range(3), [22,33,77], align='center', alpha=0.4)
plt.yticks(range(3), ["A","B","C"]) # counts.values())
plt.xlabel('counts')
plt.title('title')
plt.show()
```

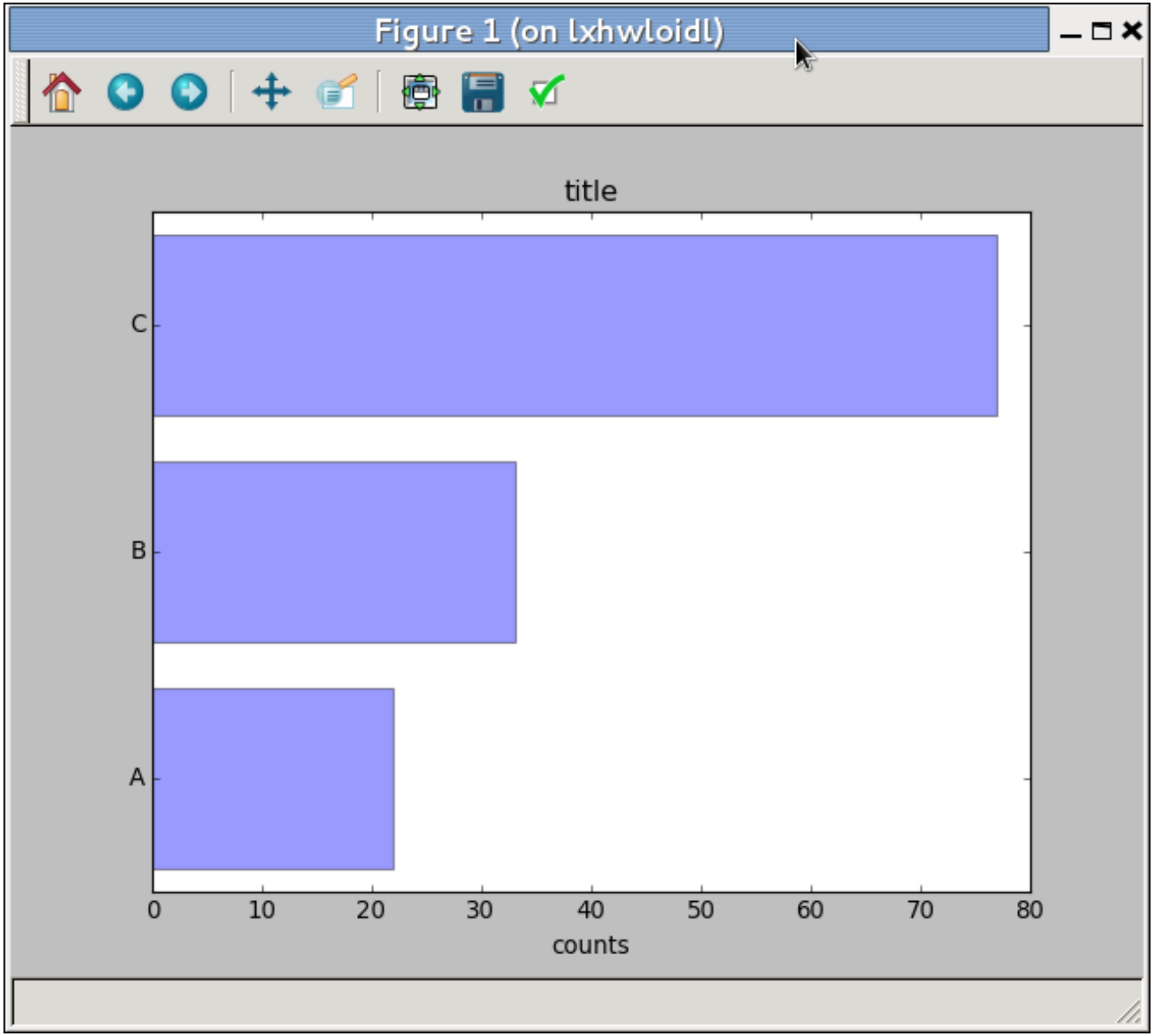
Figure 1 (on lxhwloidl)
title
C
B
A
0
10
20
30
40
50
60
70
80
counts

Examples of using `matplotlib`

Example

```
import matplotlib.pyplot as plt
...
# fixed input
counts = { 'GB' : 5, ... }
# horizontal bars: data from counts dictionary
n = len(counts)
plt.barh(range(n), list(counts.values()), align='center', a
# Beware: Python 3 ^^^^ needs a list here,
#         because counts.values() returns an iterator
plt.yticks(range(n), list(counts.keys()))
plt.xlabel('counts')
plt.title('Number of countries represented')
plt.show()
```

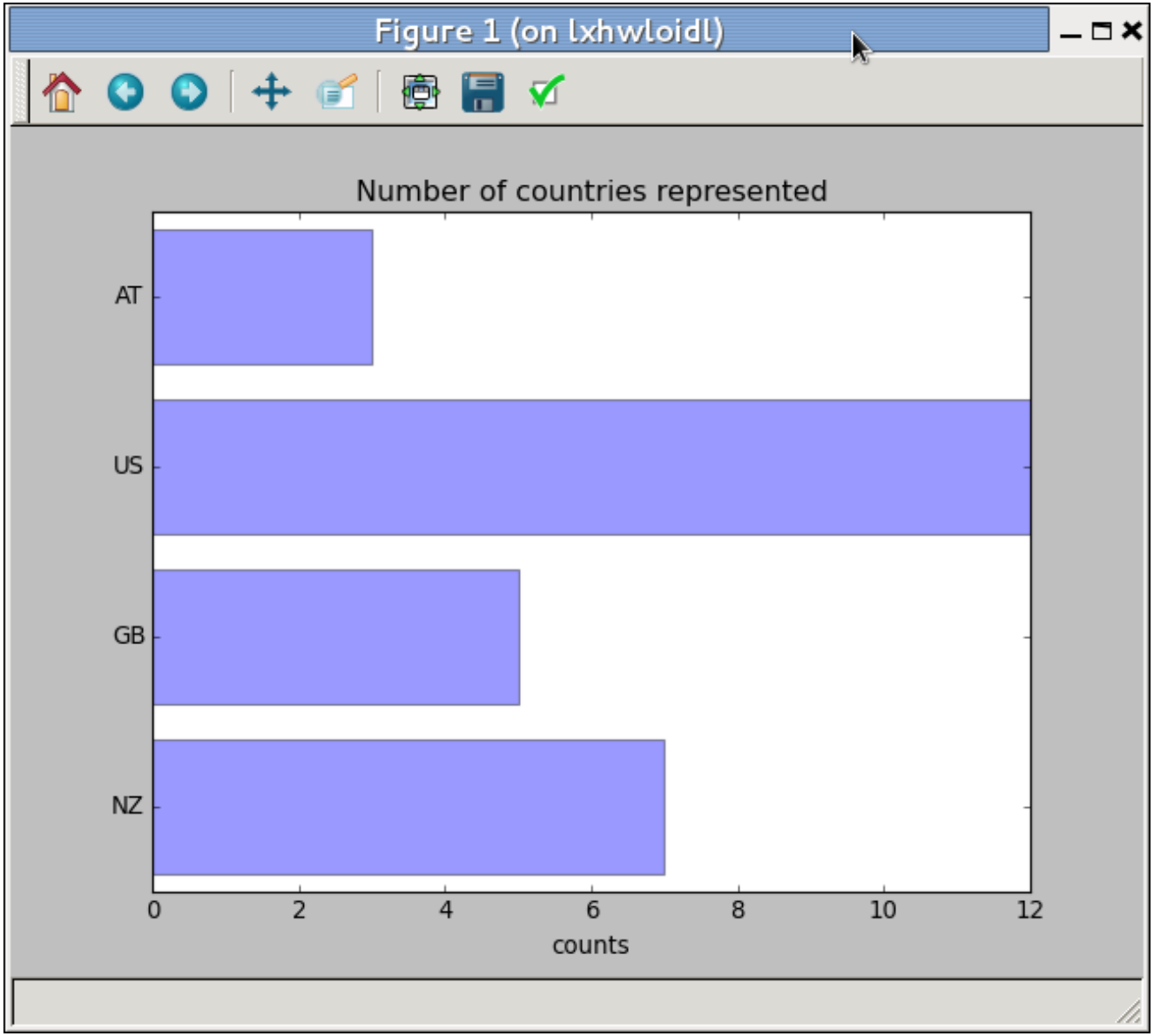
Figure 1 (on lxhwloidl)
Number of countries represented
AT
US
GB
NZ
0
2
4
6
8
10
12
counts

Examples of using `matplotlib`

A function, showing a histogram either horizontally or vertically:

Example

```
def show_histo(dict, orient="horiz", label="counts", title=
    """Take a dictionary of counts and show it as a histogr
    if orient=="horiz":    # NB: this assigns a function to
        bar_fun = plt.barh; bar_ticks = plt.yticks; bar_lab
    elif orient=="vert":
        bar_fun = plt.bar; bar_ticks = plt.xticks ; bar_lab
    else:
        raise Exception("show_histo: Unknown orientation: %
    n = len(dict)
    bar_fun(range(n), list(dict.values()), align='center',
    bar_ticks(range(n), list(dict.keys()))  # NB: uses a hi
    bar_label(label)
    plt.title(title)
    plt.show()
```

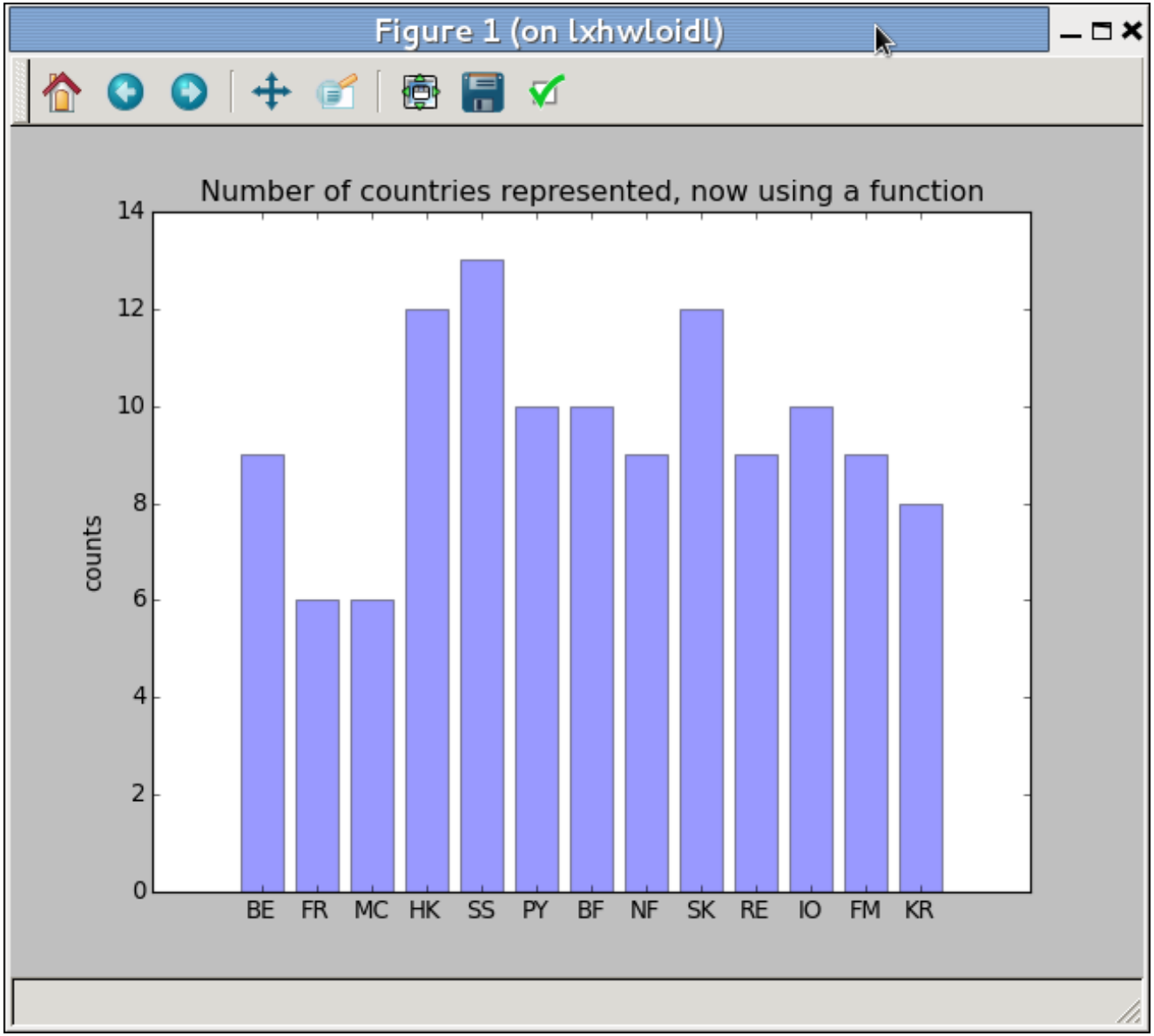
Figure 1 (on lxhwloidl)
Number of countries represented, now using a function
counts
0
2
4
6
8
10
12
14
BE
FR
MC
HK
SS
PY
BF
NF
SK
RE
IO
FM
KR

A basic GUI library for Python: `tkinter`

- `tkinter` is a basic library for graphical input/output
- It has been around for a long time, and is well supported
- It uses the Tcl/TK library as backend
- It features prominently in textbooks such as:
  Mark Lutz, *"Programming Python."* O'Reilly Media; 4 edition (10 Jan 2011). ISBN-10: 0596158106.
- For details and more examples see: tkinter documentation

For examples see Sample Sources (feet2meter.py)

Example of using `tkinter`

Example

```
from tkinter import ttk
...
root = Tk()                      # create a GUI obj
root.title("Feet to Meters") # set its title etc

mainframe = ttk.Frame(root, padding="3 3 12 12") # formatti
...

feet = StringVar()    # define a string GUI obj
meters = StringVar() # define a string GUI obj

feet_entry = ttk.Entry(mainframe, width=7, textvariable=fee
feet_entry.grid(column=2, row=1, sticky=(W, E))

ttk.Label(mainframe, textvariable=meters).grid(column=2, ro
ttk.Button(mainframe, text="Calculate", command=calculate).
```

Example of using `tkinter` (cont'd)

Example

```
ttk.Label(mainframe, text="feet").grid(column=3, row=1, sti
...

for child in mainframe.winfo_children(): child.grid_configu

feet_entry.focus()
root.bind('<Return>', calculate)

root.mainloop() # start it

#---
def calculate(*args):
    try:
        value = float(feet.get())
        meters.set((0.3048 * value * 10000.0 + 0.5)/10000.0
    except ValueError:
        pass
```

Threading

```
import threading, zipfile
class AsyncZip(threading.Thread):
    def __init__(self, infile, outfile):
        threading.Thread.__init__(self)
        self.infile = infile
        self.outfile = outfile
    def run(self):
        f = zipfile.ZipFile(self.outfile, 'w', zipfile.ZIP_
        f.write(self.infile)
        f.close()
        print('Finished background zip of:', self.infile)
background = AsyncZip('mydata.txt', 'myarchive.zip')
background.start()
print('The main program continues to run in foreground.')
background.join()    # Wait for the background task to fini
print('Main program waited until background was done.')
```

Computational Mathematics and Statistics

Sage is a free open-source mathematics software system licensed under the GPL

- It supports many computer algebra systems: GAP, Maxima, FLINT, etc
- It supports other powerful scientific engines: R, MATLAB, etc
- It includes many Python libraries for scientific computing: NumPy, SciPy, matplotlib, etc
- Python is used as **glue-ware**, all the (heavy) computation is done in the external libraries.

Example Sage Session doing Symbolic Computation

Example

```
sage: f = 1 - sin(x)^2
sage: integrate(f, x).simplify_trig()
  1/2*sin(x)*cos(x) + 1/2*x
sage: print maima(integrate(f, x).simplify_trig())
                                  cos(x) sin(x)   x
                                  ------------ + -
                                        2         2
sage: f.differentiate(2).substitute({x: 3/pi})
2*sin(3/pi)^2 - 2*cos(3/pi)^2
sage: print maima(f.differentiate(2).substitute({x: 3/pi})
                                    2  3          2  3
                              2 sin (---) - 2 cos (---)
                                    %pi           %pi
```

Numerical Computation using the `numpy` library

- `numpy` provides a powerful library of mathematical/scientific operations
- Specifically it provides
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number capabilities
- For details see: numpy documentation

Numerical Computation Example: `numpy`

Example

```
import numpy as np
m1 = np.array([ [1,2,3],
                [7,3,4] ]); # fixed test input
# m1 = np.zeros((4,3),int); # initialise a matrix
r1 = np.ndim(m1);         # get the number of dimensions for
m, p = np.shape(m1);      # no. of rows in m1 and no. of cols
# use range(0,4) to generate all indices
# use m1[i][j] to lookup a matrix element

print("Matrix m1 is an ", r1, "-dimensional matrix, of shap
```

SymPy: a Python library for symbolic mathematics

SymPy: a Python library for **symbolic mathematics**.

pandas: powerful Python data analysis toolkit

pandas is a powerful Python data analysis toolkit.

- It provides functions for constructing frames that can be accessed and manipulated like data-base tables.
- This is similar in spirit to C#'s LINQ sub-language.
- The focus is on **data manipulation**, not on statistics or scientific computing (the libraries above).

Orange: a Python library for data mining

Orange is a Python library specifically for **data analytics, data visualisation and data mining.**

Further reading

- Mark Lutz, *"Programming Python."*
  O'Reilly Media; 4 edition (10 Jan 2011). ISBN-10: 0596158106.
- Wes McKinney, *"Python for data analysis"[eBook]*
  O'Reilly, 2013. ISBN: 1449323626
  Focus on libraries for data-analytics.
- Hans Petter Langtangen, *"A Primer on Scientific Programming with Python"* 4th edition, 2014. ISBN-10: 3642549586
  Focussed introduction for scientific programming and engineering disciplines.
- Drew A. McCormack *"Scientific scripting with Python."*
  ISBN: 9780557187225
  Focussed introduction for scientific programming and engineering disciplines.