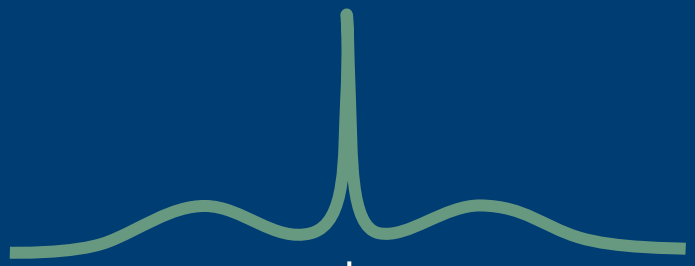
Advanced Studies School · Uberlândia

CAPES

Numerical Renormalization Group

Jan 22 – Feb 1, 2019

Main lecturer: Prof. Rok Žitko
Jožef Stefan Institute, Ljubljana, Slovenia



warm-up lecture

Computational Tools

Prof. Gerson J. Ferreira

www.infis.ufu.br/~gerson
gersonjferreira@ufu.br
[office: 1A-225]

http://www.infis.ufu.br/~escolanrg2019/

Outline

1) Intro to Linux and bash scrips (sed / grep / regex)

2) Introduction to python

→ code editors and overall syntax

3) Brief introduction to numpy and matplotlib

→ main features and commands

→ input/output

→ exercise: data smoothing

4) Data representation and multiple precision

→ Python: mpmath [equiv. in C: GMP & MPFR]

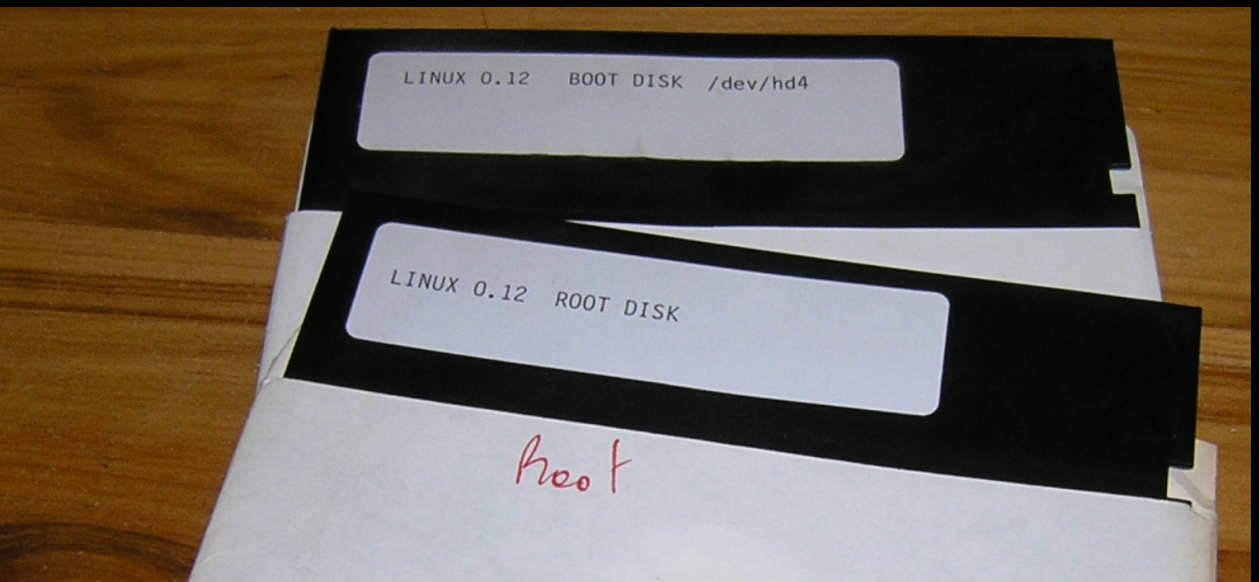
** see also my "book" on

Introduction to Computational Physics (with Julia)

http://www.infis.ufu.br/~gerson/

What is Linux?

“Microsoft isn't evil, they just make really crappy operating systems”
– Linus Torvalds (2013-01-29)



Some useful links

→ https://www.tutorialspoint.com/ ←

Linux tutorials: (includes bash, sed, awk, regex)
http://ryanstutorials.net/
https://linuxjourney.com/
http://www.grymoire.com/Unix

Bash script:
https://linuxconfig.org/bash-scripting-tutorial
http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html
http://matt.might.net/articles/bash-by-example/

Perl: https://www.tutorialspoint.com/perl/

Sed: http://www.grymoire.com/Unix/Sed.html
http://sed.sourceforge.net/sed1line.txt
https://www.tutorialspoint.com/unix/unix-regular-expressions.htm

Awk: https://likegeeks.com/awk-command/

Regular expressions (regex): https://regexone.com/
https://regex101.com/
https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285

UNIX filosofy: minimalism!

→ Doug McIlroy (1978), Peter H. Salus (1994)

Software must...

... perform a single task!
→ and do it efficiently

... work together
→ communication via I/O (fs, text, pipes)

...use the shell to combine commands
→ and perform complex tasks

GNU, Free Software Fundation

→ R. Stallman (1983)
→ promotes the universal freedom to study, distribute, create, and modify computer software

Hello everybody out there using minix - 25/agosto/1991

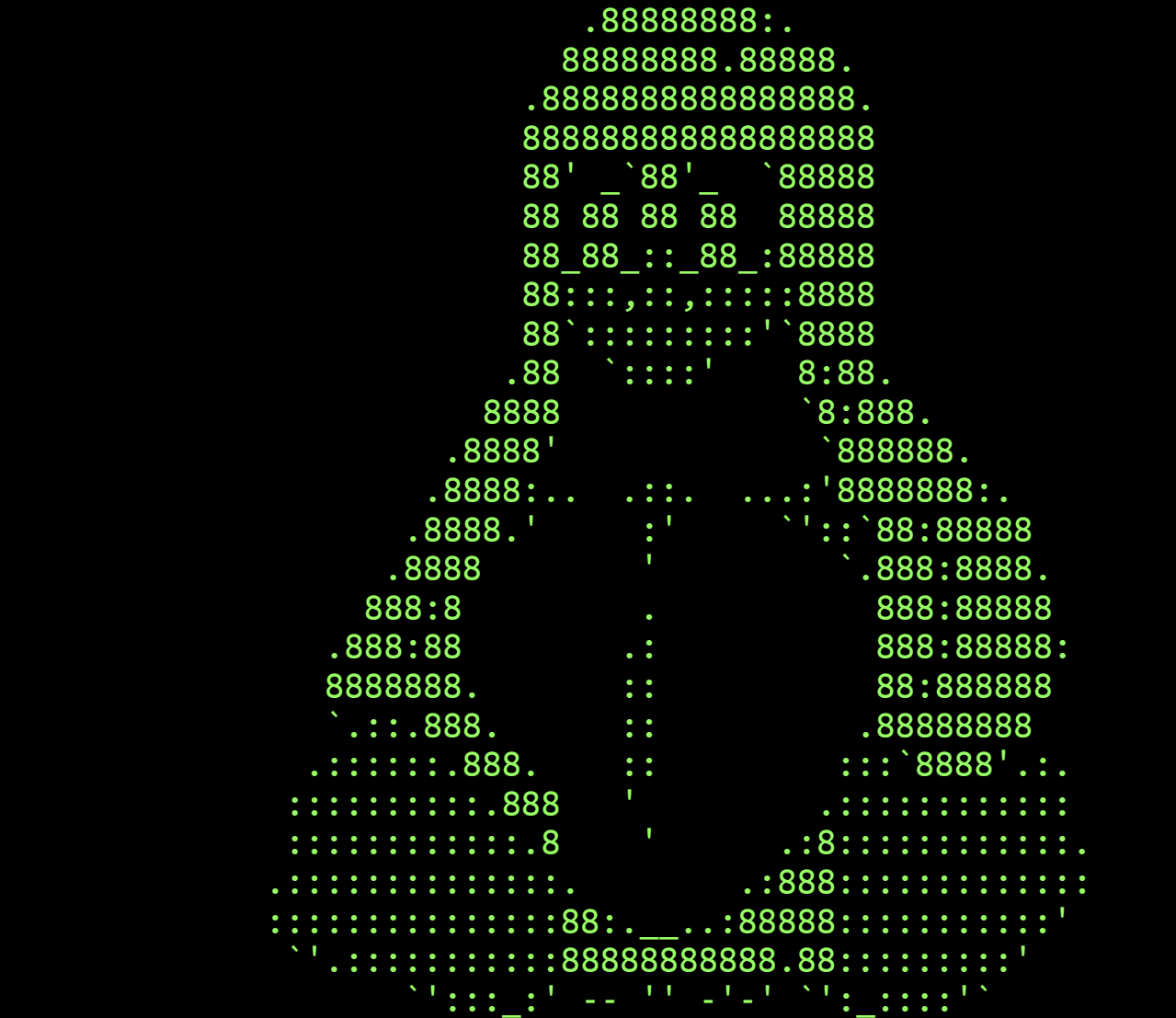
I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torv...@kruuna.helsinki.fi)

PS. To make things really clear - yes I can run gcc on it, and bash, and most of the gnu [bin/file] utilities, but it's not very debugged, and the library is really minimal. It doesn't even support floppy-disks yet. It won't be ready for distribution for a couple of months. Even then it probably won't be able to do much more than minix, and much less in some respects. It will be free though (probably under gnu-license or similar)

BASH - Bourne-Again SHell



Moving around

→ Open the terminal (bash), to access the command prompt ($)
→ It opens at the users $HOME folder.
  → check where you are by running $ pwd
  → this is your folder (user/group)
  → run $ ls to list the files at your $HOME
  → but there can be hidden files: run $ ls -a
    → files that start with . are “hidden” (config files)

→ Let’s go to /var/log
  $ cd /var/log
  $ pwd
  $ ls

→ Special characters and relative path:
  .. refers to the parent folder. Run $ cd .. and $ pwd
  ~ is a shortcut to $HOME. Run $ cd ~ and $ pwd
  . is the current folder. Useful to copy files, etc…
    example: $ cp /etc/ssh/ssh_config .

→ Tip:
  Use TAB to autocomplete commands and file names

Complex file names

→ File names are case senstive
→ Avoid unsual characters !@#$&*
→ It is better to use only
    letters [a..z]
    numbers [0..9] and
    simple symbols - _ .

→ extensions (.txt, .dat, .png) are optional, only for organization

→ Spaces are allowed, but be careful:

Try it:

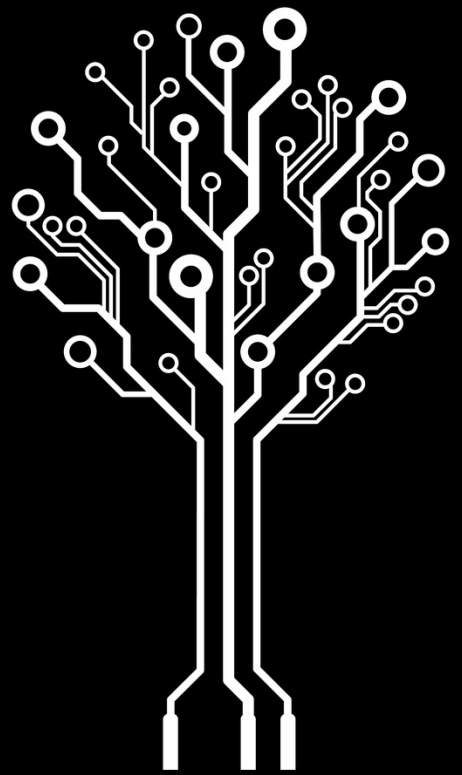
```
$ mkdir Folder with long name
$ ls -l

$ mkdir “Better use quotes”
$ ls -l

$ mkdir This\ slash\ notation\ is\ terrible
$ ls -l
```

System tree

/ - system root
/home - home folders for all normal users
/media - external devices (pendrive, cd, floppy??, ...)
/etc - system config files and init scripts
/dev - devices… within Linux, everything is a file
/var - system logs, caches
/usr - main folder for installed software
/bin - core binary files
/sbin - system binary files (config, admin)
/root - superuser home folder
/lib - libraries
/boot - kernel and boot (grub, efi, ...)
/mnt - aux folder for devices or external mounts
/opt - aux folder to store extra softwares
/proc - system and processes information
/tmp - temporary files
...



Permissions

→ Categories

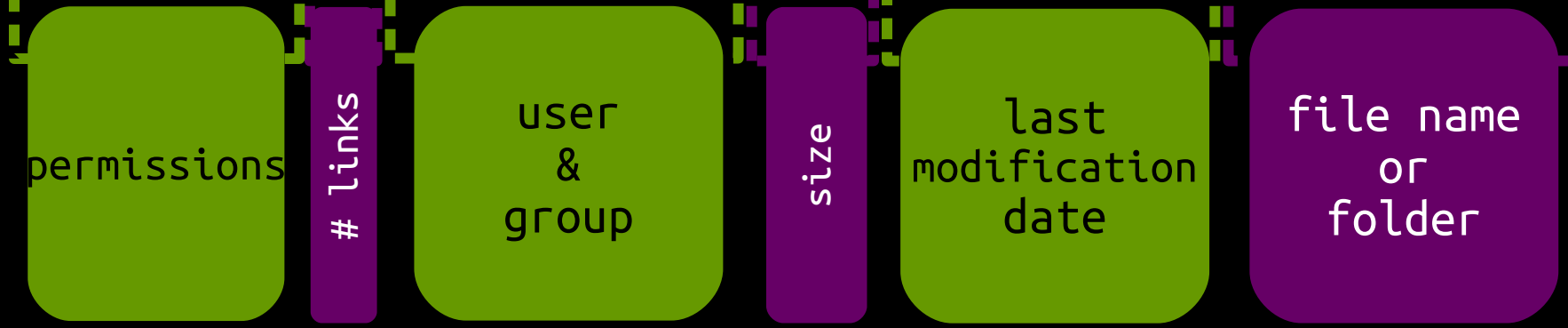
u = user, g = group, o = others

Who are you? Who, who, who, who? Run: `$ whoami` and `$ groups`

→ Let's create a file and check its permissions:

```
$ echo "Write something creative here" > myfile.txt
$ ls -l
```

```
total 32K
-rw-rw-r--  1 gerson users    18 Set 24 15:44 myfile.txt
drwxr-xr-x  2 gerson users 4,0K Set 22 11:03 Desktop
drwxr-xr-x  5 gerson users 4,0K Set 23 18:59 Documents
drwxr-xr-x  5 gerson users 4,0K Set 24 15:24 Downloads
drwx------ 35 gerson users 4,0K Set 22 09:02 Dropbox
drwxr-xr-x  8 gerson users 4,0K Set  9 07:57 public_html
drwx------ 16 gerson users 4,0K Jul 21 22:25 Sync
drwxrwxr-x  4 gerson users 4,0K Set 21 20:45 tmp
```



Permissions

```
-rw-rw-r--  1 gerson gerson   18 Set 24 15:44 myfile.txt
drwx------ 35 gerson gerson 4,0K Set 22 09:02 Dropbox
drwxr-xr-x  8 gerson gerson 4,0K Set  9 07:57 public_html
```

→ read it as: d | rwx | r-x | r-x
u g o

→ First field:
“d” indicates a directory/folder
“-” indicates a file

→ Other fields refer to the user/group/others permissions
“r” permission to read
“w” permission to write/modify/delete
“x” permission to run (exec)

→ Changing permissions
$ chmod go-rw myfile.txt ← removes(-) rw from g and o

→ Changing the ownership
$ chown root:gerson myfile.txt ← owner=root, group=gerson

Man pages

→ We'll go through many commands
→ To check the details on how to use each command...
read the manual!! ;-)

$ man man (read the manual on how to use the manual)

→ to navigate: arrows
→ to search: / , n (next)
→ to quite: q
→ help: h

Example:

$ man grep
→ search por "case"

→ check also the command whatis:

$ whatis cp
$ whatis grep

Files: creating/removing/copying/...

→ Create a blank file

```
$ touch blank.txt
$ ls -l
```

→ Create some folders

```
$ mkdir My_Files Another_Folder
```

→ Move blank.txt to My_Files

```
$ mv blank.txt My_Files
$ ls My_Files
```

→ Renaming files or folders is the same as moving

```
$ mv My_Files New_Name
$ ls
```

→ Using relative paths:

```
$ cd Another_Folder
$ mv ../New_Name .
```

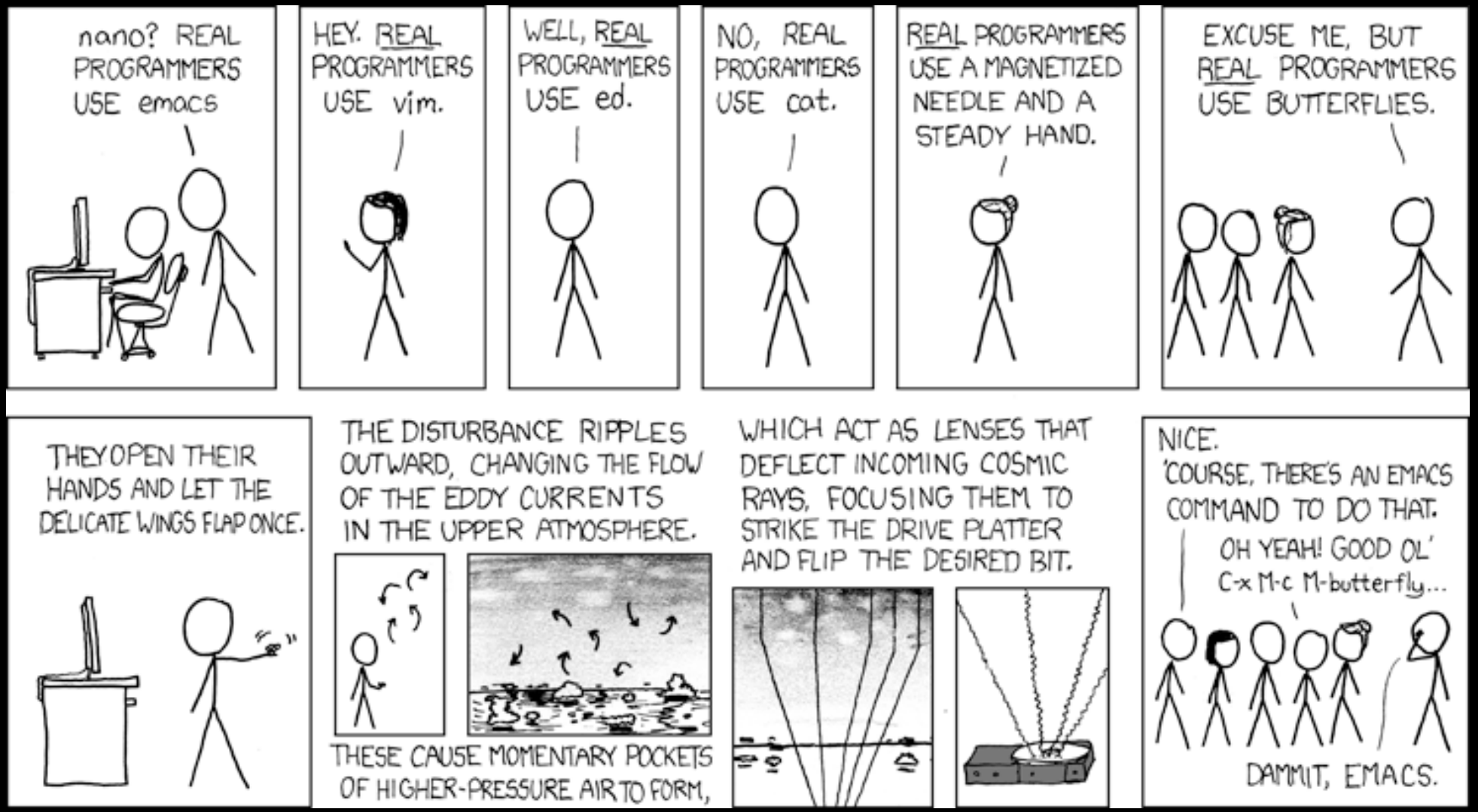
→ Let’s delete all files and folders we have created here

```
$ rm <file1> <file2> …
$ rmdir <dir1> <dir2> …
```

File editors

→ Main editors: vim, emacs

... both complicated, both powerful!!



Real programmers set the universal constants at the start such that the universe evolves to contain the disk with the data they want.

File editors

minimalist set of commands

emacs

```
$ emacs myfile.txt

Save  →  CTRL+x CTRL+s
Quit  →  CTRL+x CTRL+c
Abort →  CTRL+g
```

More:
https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf

vim

```
$ vim myfile.txt

Edit     → i
Commands → ESC

Save     → :w
Quit     → :q
Force    → :q!
```

More:
http://tnerual.eriogerg.free.fr/vimqrc.pdf

nano

```
$ nano myfile.txt

Save  →  CTRL+o
Quit  →  CTRL+x
```

gedit

```
$ gedit myfile.txt

(graphical interface)
```

Quickly reading files

→ cat (tac) prints (stnirp) the file on screen:

```
$ cat myfile.txt
```

→ less: show file interactively

```
$ less myfile.txt (arrows, /=search, h=help, q=quit)
```

→ `head -n 3 myfile.txt` : prints the first 3 lines

→ `tail -n 3 myfile.txt` : prints the last 3 lines

→ grep: prints lines that match a pattern (see regex)

Try it:

```
$ grep --color Nobel nrg.txt
```

Input / output redirect

```
$ command > file.txt  : redirect output to file
$ command >> file.txt : append into file

$ command 1> file.txt 2> errors.txt
          redirects main output to file.txt
                    error output to errors.txt

$ command_1 | command_2 : |=pipe captures output from 1
                          and redirects as the input of 2
```

Examples (check the commands without redirection as well)

```
$ echo “Hello world” > hello.txt
$ ls -l
$ cat hello.txt

$ echo “one more line” >> hello.txt
$ cat hello.txt

$ ls / | grep bin
$ ps -ef | grep bash
```

Scripts

→ Scripts start with #! = hash(#) bang(!) = hashbang
  Indicates the language, which must work with pipes |

  → bash: #!/bin/bash
  → gnuplot: #!/usr/bin/gnuplot
  → python: #!/usr/bin/python3
  → perl: #!/usr/bin/perl

  Examples:

→ Edit hello.sh, and give it permission to run $ chmod +x hello.sh
→ To run, call $ ./hello.sh

```
#!/bin/bash
myname=`whoami`
echo "Hello world! My name is $myname."
```

→ Now in Perl:

```
#!/usr/bin/perl
$myname=`whoami`;
print("Hello world! My name is $myname.\n");
```

Gnuplot example

see http://gnuplot.sourceforge.net/demo/
my old gnuplot class in www.infis.ufu.br/~gerson

→ Example 1: plot on screen

```
#!/usr/bin/gnuplot --persist

set xlabel('T')
set ylabel('S_z')
# from file termo.dat, plot using x = column 1, y=column 2
# with lines and points and point type = circles (7)
plot 'termo.dat' u 1:2 w lp pt 7
```

→ Example 2: plot to file

```
#!/usr/bin/gnuplot --persist
set terminal png size 800,600 nocrop
set output "gpfigure.png"

set xlabel('T')
set ylabel('S_z')
plot 'termo.dat' u 1:2 w lp pt 7
```

Scripts

→ good scripts require knowledge of the linux commands

#! - hashbang, first line, defines the language

cd - change directory
ls - list content of directory
pwd - print working directory
mkdir - make directory
rm - remove / delete
rmdir - remove / delete folders (directories)
mv - move / rename

cat - print file on screen (see tac as well)
echo - print text or variables on screen
less - show file on screen interactivelly
grep - capture lines accordingly to pattern (see regex)

kill - signals a process, usually to kill it

redirecting: > (write) >> (append) < (read) | (pipe)

Scripts

→ many… many useful commands (all work with files or pipes)

head - print the first lines
tail - print the last lines
sort - sort lines (check $ man sort)

nl - count number of lines
wc - count characters, words, paragraphs...

cut - cut sections
paste - merge sections
join - join lines

uniq - remove repeated lines
tr - translate or delete characters

egrep - grep + regular expressions (regex)
regex - advanced pattern matching

sed - stream editor to transform / manipulate text
awk - pattern processing language

...

Example: simple backup

→ Create backup.sh:

```
#!/bin/bash

# save backup to
bakdir="$HOME/backup"
echo "Backup directory: $bakdir"

# creates folder if necessary
mkdir -p $bakdir

# today's date on ISO8601 format YYYY-MM-DD
today=`date -I`

# backup file name
fname="$bakdir/$hoje.tgz"

# compacts the current directory and save it
# in the backup folder
tar -czf $fname ./*
```

Example: special vars

→ Edit args.sh:

```
#!/bin/bash

echo "Script name: $0"
echo "Number of parameters: $#"
echo "Parameter array: $@"

id=1
for par in $@; do
    echo "Parameter $id = $par"
    let id++
done
```

→ Arithmetic operations in bash require

(i) the macro let

```
let id++
let id=id+1
```

(ii) or the double parenthesis:

```
(( id++ ))
id=$((id+1))
(( id=id+1 ))
```

Examples: cut and paste columns

→ Edit cutpaste.sh:

```
#!/bin/bash

# using redirect to merge one under the other
files=`ls file*.txt`
new="merged.txt"
echo -n "" > $new
for eachfile in $file; do
    cat $eachfile >> $new
done
```

→ run these directly on bash

```
# merge all columns side by side
paste -d ' ' file1.txt file2.txt > pasted.txt

# extracts some columns
cut -d ' ' -f 1,3 pasted.txt > cut.txt

# equivalently:
paste -d ' ' file1.txt file2.txt | cut -d ' ' -f 1,3 > cut2.txt
```

Using sed

→ Check the file Linux_win.txt. Clearly wrong! Let's fix it!

→ Replace first occurrence on each paragraph

```
$ sed 's/windows/linux/' Linux_win.txt
```

→ Replace n-st occurence in each paragraph

```
$ sed 's/windows/linux/2' Linux_win.txt
```

→ Replace all occurences

```
$ sed 's/windows/linux/g' Linux_win.txt
```

→ Run and try to explain these commands to me:
check $ man sed if necessary

```
$ sed 's/windows/!&?!/g' Linux_win.txt ← try also with &&

$ sed -i 's/windows/{aux}/g; s/linux/windows/g;
                          s/{aux}/linux/g' Linux_win.txt
```

→ great! now it looks good!!

Regular expressions (regex)

Can be used with bash, perl, python, C, …

Let’s use the file cordel.txt in our examples:

```
# print lines that contain ‘capaz’
grep --color capaz cordel.txt

# print lines that start (^) with ‘Eu’
grep --color ^Eu cordel.txt

# print lines that end ($) with ‘;’ (special character)
grep --color \;$ cordel.txt

# print lines that cotain ‘sã’ or ‘nã’
grep --color [sn]ã cordel.txt

# print lines that start with A, B, C, D
grep --color ^[A-D] cordel.txt

...
```

Introduction to Python

Overall syntax and Code editors

Python Code Editors: my favorites

ATOM
+ Hydrogen package
[www.atom.io]

nteract
[www.nteract.io]

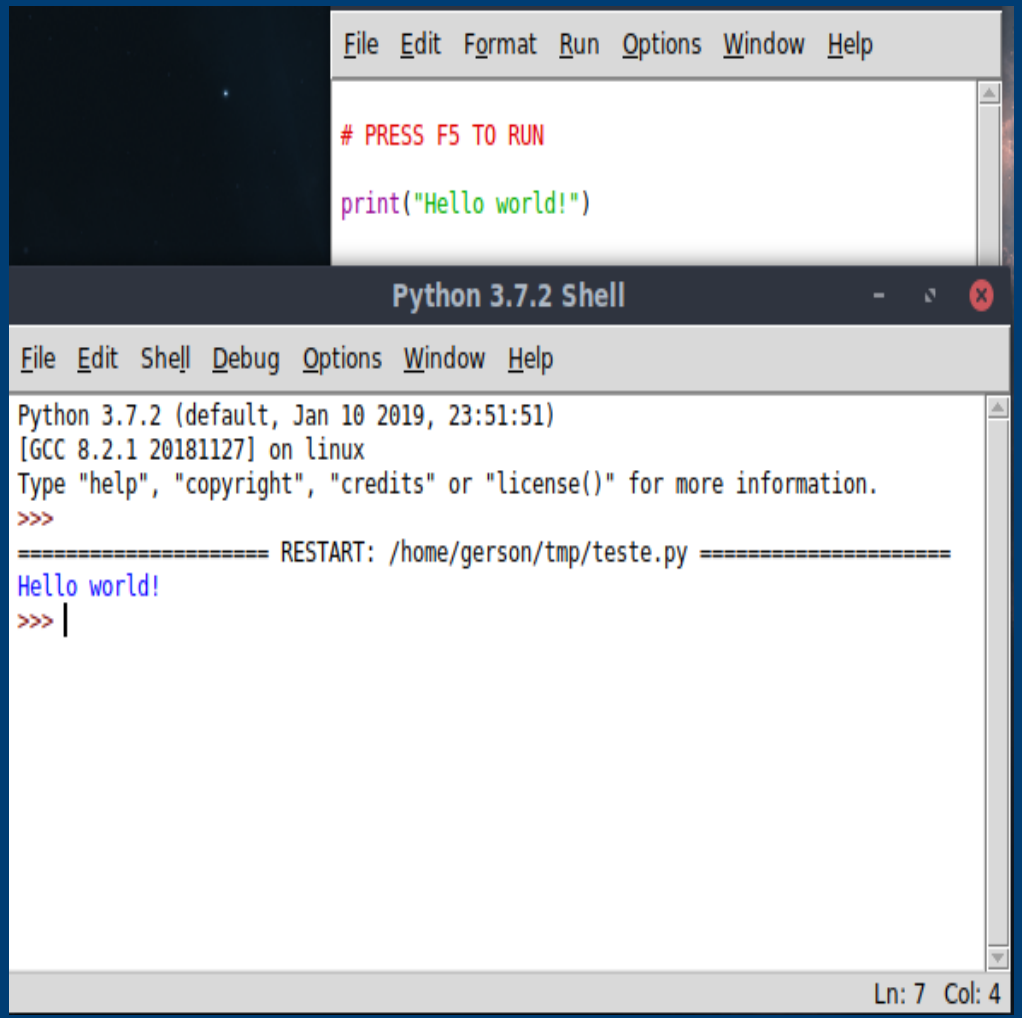
[www.jupyter.org]

JupyterLab
[https://github.com/jupyterlab/jupyterlab]

Python Code Editors: available in this class

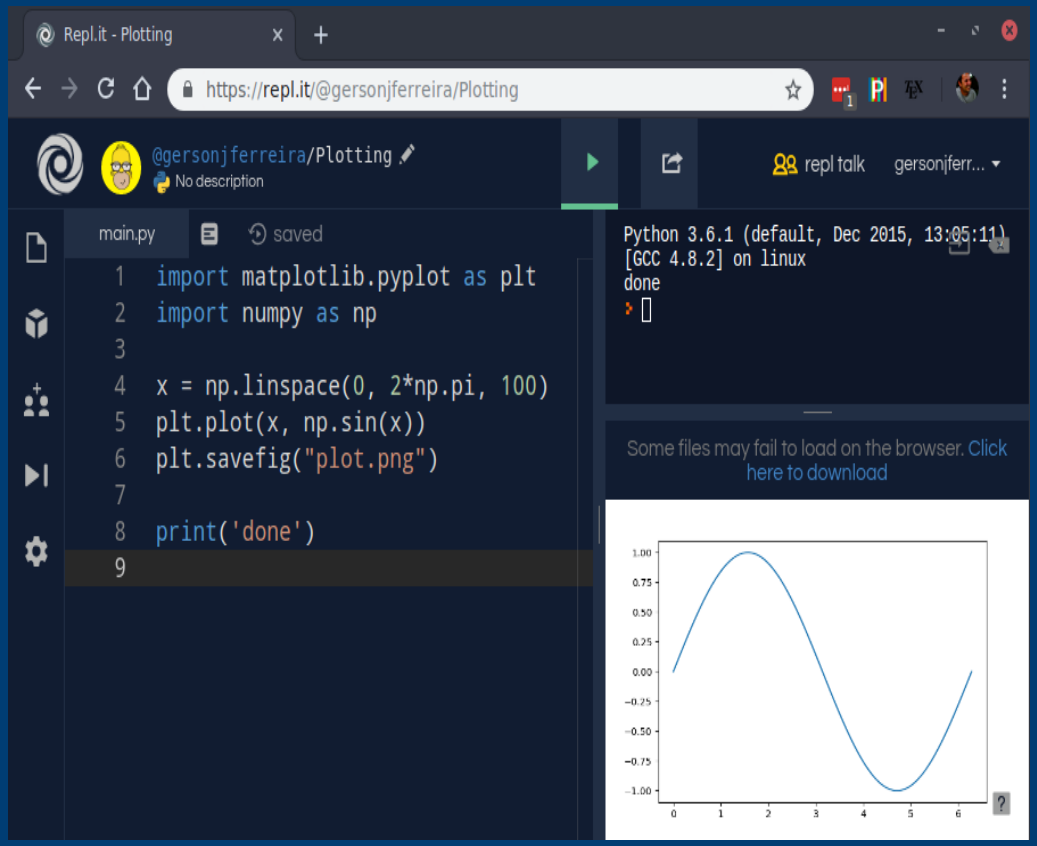
IDLE: simple, but useful

→ make sure to open the Python 3 version!



repl.it

Online: www.repl.it



Python tutorial

[https://www.w3schools.com/python/python_intro.asp]

Python: basic operations

1) Assignment

```
a = 3
b = 4
c = a**2 + b**2
print('c = ', c)
```

2) if statement

```
if a > b:
   print('a is larger than b')
elif a < b:
   print('b is larger than a')
else:
   print('a is equal to b')
```

3) types of variables

```
x = 5     # int
y = 4.2 # float
Na = 6.022e23 # float
z = 3 + 4j # complex
a = 'hello' # string

print(type(z))
...
```

4) operators

```
Arithmetic: + , - , * , /
Exponentiation: ** ,
   ex: 1024 == 2**10
Assignment: = , += , -= , *= , /=
   ex: x *= 3 → x = x*3
Comparison: ==, !=, >, >=, <, <=
Logical: and , or , not
Membership: in , not in
   ex: 5 in [3, 2, 9] → False
```

Python: basic operations

5) Lists (for vectors → see numpy)

```
fruits = ['banana', 'orange']
fruits.append('apple')
fruits.sort()
print('Length:', len(fruits))
print('first:', fruits[0])
print('last:', fruits[-1])
```

6) For loop over lists

```
for fruit in fruits:
    print('we have ', fruit)
```

7) Membership

```
if 'grape' in fruits:
    print('yes, we have grapes')
else:
    print('no, we don't have grapes')
```

8) Lists over range of integers

```
syntax:
    range(n) → 0,1,..n-1
    range(i, f, s)
        i → initial
        f → final, not included
        s → step

for i in range(3, 15, 2):
    print(i)

for i in range(len(fruits)):
    print(i, fruits[i])
```

Python: basic operations

9) Functions

```
# import single function
# from the math library
from math import sqrt

# define the function
# here c has a default value
def bhaskara(a, b, c=0):
    d = sqrt(0j+b**2-4*a*c)
    x1 = (-b+d)/(2*a)
    x2 = (-b-d)/(2*a)
    return x1, x2

# calling the function
s1, s2 = bhaskara(1, -5, 6)
print('sols:', s1, 'and', s2)

# check with bhaskara(1, -5, 0)
```

10) Lambda inline functions

```
from math import exp, sin

# short function can be written
# in a compact notation
sinc = lambda x: sin(x)/x
gauss = lambda x: exp(-x**2)

y = sinc(1.0)*gauss(3.0)

print('y = ', y)
```

Python: basic operations

11) Libraries

```
# to import a single function
from math import sqrt
print(sqrt(2))
help(sqrt) # help only works with imported objects

# to import everything
# not recommended → possible conflicts
from math import *
print(cos(pi))

# import under a short name
import math as m
print(m.sin(m.pi/2))
help(m)
```

Instead of the math library, we will typically prefer to use numpy:

→ includes math, num. calculus, linear algebra, ...

```
import numpy as np
print(np.cos(np.pi))
```

Python: basic operations

→ Comprehensions : simple form to create lists and matrices (+ numpy)

```
mylist = np.array([ f(x) for x in xlist ])
Nested lists form matrices:
mymatrix = np.array([ [g(x,y) for x in xlist] for y in ylist ])
```

→ Dictionaries : very useful to store/organize input parameters

```
params = {}
params['temperature'] = 300
params['mag. field'] = 10
params['method'] = 'RK4'

print(params)
print('T = ', params['temperature'])
```

Exercise: factorial

- Create a function called "myfactorial" that receives a number n and returns the factorial n!.
- Do not use recursive functions! Implement the factorial using a for loop.
- Compare the result with the native factorial function from the math library.
- Test it with the code below:

```
from math import factorial

def … # implement your function here

# testing:
n = 10
print('Exact: ', factorial(n))
print('Mine: ', myfactorial(n))
```

Answer:

Expected result: 3628800

Intro to numpy and matplotlib

reading/saving files, manipulating data, plotting

[https://www.tutorialspoint.com/numpy/index.htm]

[https://www.tutorialspoint.com/python/python_matplotlib.htm]

Python + numpy : vector-based language

The main element in numpy are the nd-arrays (nd = n-dimensional)

→ vectors [= vector/matrix/tensor]

```
import numpy as np

v1 = np.array([7, 4, 9, 10]) # 4-component vector
v2 = np.linspace(5, 12, 4) # (initial, final*, step)
print('v1 = ', v1, ', v2 = ', v2)

d = np.dot(v1, v2) # nd-scalar product
print('scalar product = ', d)

x = np.array([1, 0, 0])
y = np.array([0, 1, 0])
z = np.cross(x, y) # cross product only for 2D and 3D arrays
print('z = ', z)
```

Python + numpy : vector-based language

→ matrices

```
import numpy as np

mat1 = np.array([ [1,2] , [3,4] ]) # a 2x2 matrix

line1 = np.array([1,2]) # each line is a vector
line2 = np.array([3,4])

mat2 = np.array([ line1, line2 ]) # another way to write mat1

print('matrix 1 = ', mat1)
print('matrix 2 = ', mat2)
```

Python + numpy : vector and matrix operations

vectors

```
v1 = np.array([1,2,3])
v2 = np.array([4,5,6])

print(v1+v2)
print(v1*v2)
print(v1**2)
print(np.sin(v1*np.pi))

v3 = np.zeros(5)
v4 = np.ones(5)

print(v3)
print(v4)
```

matrices

```
m0 = np.zeros((5,5))
m1 = 2*np.diag(np.ones(5))
m2 = -np.diag(np.ones(4), +1)
m3 = -np.diag(np.ones(4), -1)
m4 = m1 + m2 + m3

print(m4) # Exercise: explain the commands

v0 = np.array([1, -1, 0, 1, -1])

print(m4*v0) # what is the difference
print(np.dot(m4, v0)) # between these two?
```

Python + numpy : vector and matrix elements

→ array indexes start from 0

vectors

```
v1 = np.array([6, 3, 4, 2])

print(v1[0]) # to access an specific element
print(v1[-3]) # can count backwards, -1=end, and so on

v1[-3] = 5 # or change its value
print(v1)

print(v1[1:3]) # open interval (i:f(
print(v1[:3]) # i=0 if omitted
print(v1[1:]) # f=-1 if omitted
print(v1[0:-2]) # counting backwards
```

Python + numpy : vector and matrix elements

matrices

```
m1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(m1)
print(m1[1,2])
# line 1, column 2 : recalling that starts from 0

print(m1[:,2])
# [:,2] = [0:-1,2] = all lines, column 2

print(m1[1,:]) # line 1, all columns

print(m1[1:, 1:]) # extracting a submatrix
```

Python + numpy : linspace vs arange

Important, similar, and confusing ways of defining a discrete axis

np.linspace

(initial, final, number of points)

```
a = 5    # initial point
b = 11   # final point
N = 4    # number of points

x = np.linspace(a, b, N)
print(x)

dx = x[1]-x[0]
```

np.arange

(initial, final*, step)

```
a = 5    # initial point
b = 11   # final* point
dx = 2   # step size

x = np.arange(a, b+dx/2, dx)
print(x)

N = len(x)
```

* open interval

Matplotlib: simple plots

Let's start with some basic plots

```
import numpy as np
import matplotlib.pyplot as plt # standard call
plt.rc('font', size=22) # overall font size

k_rng = np.linspace(-50, 50, 100) # elements ( k[i], ek[i] )
ek_rng = 0.5*k**2                 # form ordered pairs for the plot

plt.figure(figsize=(6,4)) # figure size in inches [1 in = 2.54 cm ]

#        x   y
plt.plot(k_rng, ek_rng, lw=3, color='red')

plt.xlabel(R'$k$') # R indicates Latex
plt.ylabel(R'$\varepsilon(k)$')
plt.tight_layout() # recenter figure to the visible window
plt.savefig('parabola.png') # relative to running path
plt.show() # to show on screen
```

Python + numpy : input / output

Check the file termo.dat → 8 columns

```
import numpy as np
import matplotlib.pyplot as plt
plt.rc('font', size=22)

→ data = np.loadtxt('termo.dat') # read and store as nd-array

print('lines and cols = ', data.shape)

plt.plot(data[:,0], data[:,1]) # x=col 0, y=col 1
plt.xlabel(R'$T$')
plt.ylabel(R'$\langle S_z^2 \rangle$')
plt.tight_layout()
plt.show()
```

Python + numpy : input / output

To save data in a N-columns data file, organize it as a matrix.

```
import numpy as np

theta = np.linspace(0, 2*np.pi, 13)
sin = np.sin(theta)
cos = np.cos(theta)

tosave = np.array([theta, sin, cos]) # lines
print('shape = ', tosave.shape)

→ tosave = np.array([theta, sin, cos]).T # transpose to columns
  print('shape = ', tosave.shape)

→ np.savetxt('out.dat', tosave)
```

Exercise: 1D density of states ↔ data smoothing

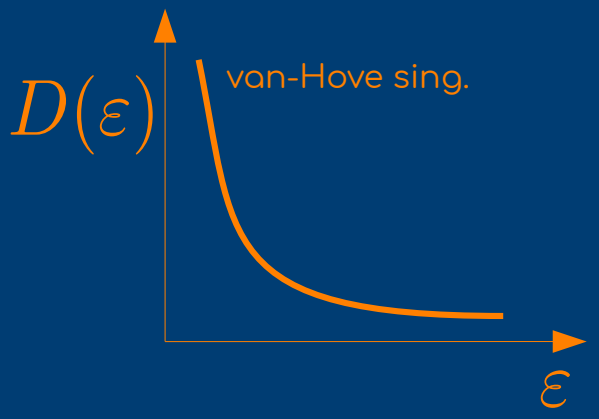
From Solid State Physics:

$$D(\varepsilon) = \frac{1}{L}\sum_k \delta(\varepsilon - \varepsilon_k) \quad \text{such that} \rightarrow \quad n_{1\mathrm{D}} = \frac{N}{L} = \int_0^{\varepsilon_F} D(\varepsilon) d\varepsilon$$

If we know the dispersion exactly : $\varepsilon_k = \frac{1}{2}k^2$

we can proceed to obtain :

$$D(\varepsilon) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} \delta(\varepsilon - \varepsilon_k) dk = \frac{1}{\pi\sqrt{2\varepsilon}}$$



But instead, let us replace delta → gaussian, and sum numerically

$$D(\varepsilon) \approx \frac{\Delta k}{2\pi}\sum_k g(\varepsilon - \varepsilon_k) \qquad g(\varepsilon) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left[-\frac{1}{2}(\varepsilon/\sigma)^2\right]$$

Exercise: 1D density of states

$$D(\varepsilon) \approx \frac{\Delta k}{2\pi} \sum_k g(\varepsilon - \varepsilon_k) \qquad \varepsilon_k = \frac{1}{2}k^2 \qquad g(\varepsilon) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}(\varepsilon/\sigma)^2\right]$$

---

1) From the matplotlib example, we already have
   k → k_rng, and ε_k → ek_rng

2) From k_rng, calculate Δk (see linspace)

3) Define a function gauss(ε, s), following the Gaussian definition above
   → later we will use s → σ=10

4) Use linspace to define an energy range from -50 to 1000 with 1000 points

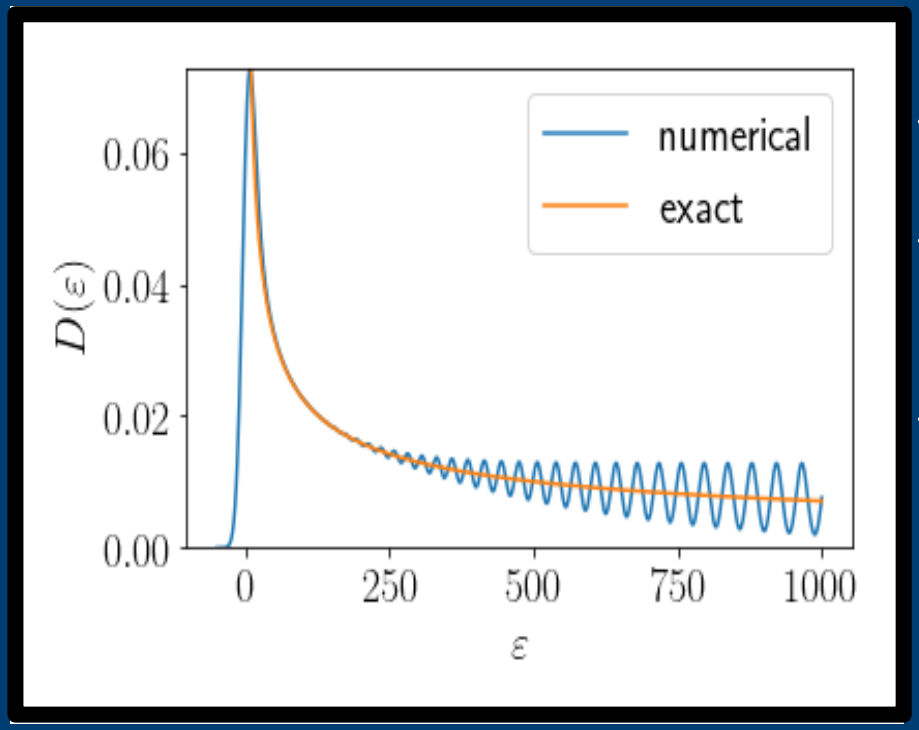
5) Initialize a variable dos using np.zeros(...) with the same number of points

6) Loop over ek_rng to sum the dos, and plot D(ε) vs ε. Result → next slide

Exercise: 1D density of states

$$D(\varepsilon) \approx \frac{\Delta k}{2\pi} \sum_k g(\varepsilon - \varepsilon_k) \qquad \varepsilon_k = \frac{1}{2}k^2 \qquad g(\varepsilon) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}(\varepsilon/\sigma)^2\right]$$

7) Try to plot the final figure like this one



→ Why are there oscillations?

→ Try increasing the number of points in k_rng

→ Code will be available after the class

Numerical calculus and linear algebra

Simple examples on how to use numpy / lapack

The numpy & scipy libraries

main libraries for scientific computing in Python

```
import numpy as np
import scipy as sp
```

Main features:

→ ND Arrays = vectors, matrices, tensors

→ Linear Algebra = matrix/vector products,
eigenproblems (LAPACK), decompositions,

...

→ Fast Fourier Transforms (FFT)

→ Input / output

→ Special functions

→ Integration, differential equations, Runge-Kutta, ...

→ Interpolation

→ Symbolic mathematics

Eigenproblem example: infinite quantum well

The discrete kinetic energy is

$$H_k = \frac{1}{2}k^2 = -\frac{1}{2}\partial_x^2 = -\frac{1}{2}\begin{pmatrix} -2 & 1 & 0 & & \\ 1 & -2 & 1 & 0 & \\ 0 & 1 & -2 & 1 & 0 \\ & 0 & 1 & -2 & 1 \\ & & 0 & 1 & 2 \end{pmatrix}$$

Let's calculate its eigenvalues and eigenvectors

```
import numpy as np
import matplotlib.pyplot as plt

N = 15 # size of the matrix, well length
Hk = -2*np.diag(np.ones(N)) # diagonal
Hk += np.diag(np.ones(N-1), +1) # top subdiagonal
Hk += np.diag(np.ones(N-1), -1) # lower subdiagonal
Hk *= -1/2 # common prefactor

→ evals, evecs = np.linalg.eigh(Hk)
print('Energies: ', evals[0:3]) # first 3 energies

# trick to add the outer points (boundary condition, psi=0)
evecs_bc = np.vstack([np.zeros(N), evecs, np.zeros(N)])

plt.plot(evecs_bc[:, 0:3]) # plot first three psi
plt.show()
```

Data representation

Floating-point numbers and multiple-precision

Data representation

Real numbers

Example of exact representations:

$$123.456 = 123456 \times 10^{-3} = 1.23456 \times 10^{2}$$

significand x base $^{\text{exponent}}$

Integers

42 : what is the meaning of all this?

how does the computer store these numbers?

what are the limitations?

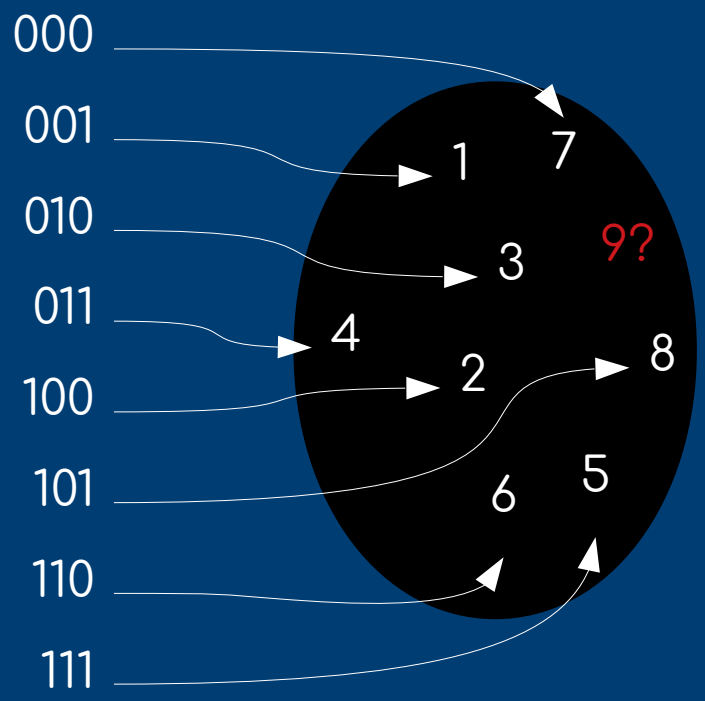
8-bits, 16-bits, 32-bits, 64-bits, 128-bits ... ?

(nes, snes, ps, n64, ps2, xbox, ... ;-)

Each CPU implements native n-bits operations [current PCs: 64-bits]

Example: it limits the memory allocation and data representation

A 3-bit memory controller
can only point to $2^3 = 8$ addresses



A 3-bit memory bank
can only represent 8 abstract entities

000 = apple
001 = avocado
010 = banana
011 = lemon
100 = pineapple
101 = strawberry
110 = kiwi
111 = grape

orange?

Integer representation

From the previous slide, a 3-bit computer can only represent 8 integers

3-bit binary memory

$= 1x2^2 + 0x2^1 + 1x2^0 = 5$

Min: (000)b = 0
Max: (111)b = 7

Signed 32-bit integer standard

Sign +/-

+ 31-bits

$= (-1)^{b31} \times (b_{30} \times 2^{30} + b_{29} \times 2^{29} + \ldots b_0 \times 2^0)$

"sign-magnitude representation"

Range 9+ digits

Min: $-2^{31} = -2{,}147{,}483{,}648$
Max: $+2^{31}-1 = +2{,}147{,}483{,}647$

"2's complement representation"

Real numbers : floating point

significand x base $^{\text{exponent}}$

Signed 32-bit float standard

exponent (8 bits)
= signed int

|significand| (23 bits)

sign of the significand

While the set of real numbers is dense / continuous...

only a limited number of entities can be represented on a finite structure

→ loss of precision

Comparing real numbers

Let us try this in Python

1) initialize the variables as

```
x = 0.1
y = 3*x
z = 0.3
```

2) check the output of the equality tests

```
z == 0.3
y == 0.3
y == 3*0.1
```

3) What is happening? Let us print x, y, z with many digits

```
print('x = ', format(x, '0.30f'))
print('y = ', format(y, '0.30f'))
print('z = ', format(z, '0.30f'))
```

Why is 0.1 = 0.1000000000000000055511151231 26 ?

If a float in simply stored as significand x base $^{\text{exponent}}$

shouldn't we have $0.1 = 1 \times 10^{-1}$?

... but computers use base 2 !!!

Examples

→ the fraction 1/10 in base 10 is 0.1 (exact)

→ the fraction 1/3 in base 10 is $0.\overline{3}$ = 0.333333... ~ 0.333334

→ the fraction 1/10 in base 2 is $0.0\overline{0011}$ = 0.0001100110011001...

An exact fraction in base 10 might be a repeating fraction in base 2

Truncation yields: $(0.0001100110011)_2 = (0.0999755859375)_{10}$

To convert from decimal to binary: [https://www.rapidtables.com/convert/number/binary-to-decimal.html]

Multiple, or arbitrary-precision

Default types:

C

[int] = 32 bits
[float] = 32 bits
[double] = 64 bits
...

Python

[int] = arbitrary
[float] = 64 bits

Allows you to freely choose the precision

Floats: 32 bits ~ 7 digits, 64 bits ~ 16 digits, 128 bits ~ 34 digits

GMP: GNU Multiple Precision Arithmetic Library

[https://gmplib.org/]

MPFR: multiple-precision floating-point computations with correct rounding

[https://www.mpfr.org/]

Python mpmath [http://mpmath.org/]

→ for multiple-precision in Python, uses GMP if available

Python SymPy [https://www.sympy.org/]

→ for Symbolic mathematics in Python

Example in Python

From "Why and How to Use Arbitrary Precision",
Ghazi et al, Computing in Science & Engineering 12, 62-65 (2010)

Let us try to calculate $d = 173746a + 94228b - 78487c$, with

$a = \sin(10^{22})$, $b = \log_{10}(17.1)$, $c = \exp(0.42)$

Standard Python

```
import numpy as np
a = np.sin(1e22)
b = np.log(17.1)
c = np.exp(0.42)
d = 173746*a + 94228*b - 78487*c
print('d =', d)
      2.9103830456733704e-11
```

Exact value:
$-1.341818958\ldots \times 10^{-12}$

The same Python example, but using mpmath

```
from mpmath import mp
mp.dps = 40 # defines the decimal precision
print(mp) # to check again

a = mp.sin('1e22')
b = mp.log('17.1')
c = mp.exp('0.42')
d = 173746*a + 94228*b - 78487*c
print('d = ', d)
      -1.34181895782961046706215258814e-12
```

Exact value:
$-1.341818958\ldots \times 10^{-12}$

Notice that the real numbers are set by strings, otherwise Python would first convert to floating-point and lose precision. TRY IT!

Overall features of these libraries

	python numpy+scipy	python mpmath	C+GMP+MPFR
Float precision	64 bits	arbitrary	arbitrary

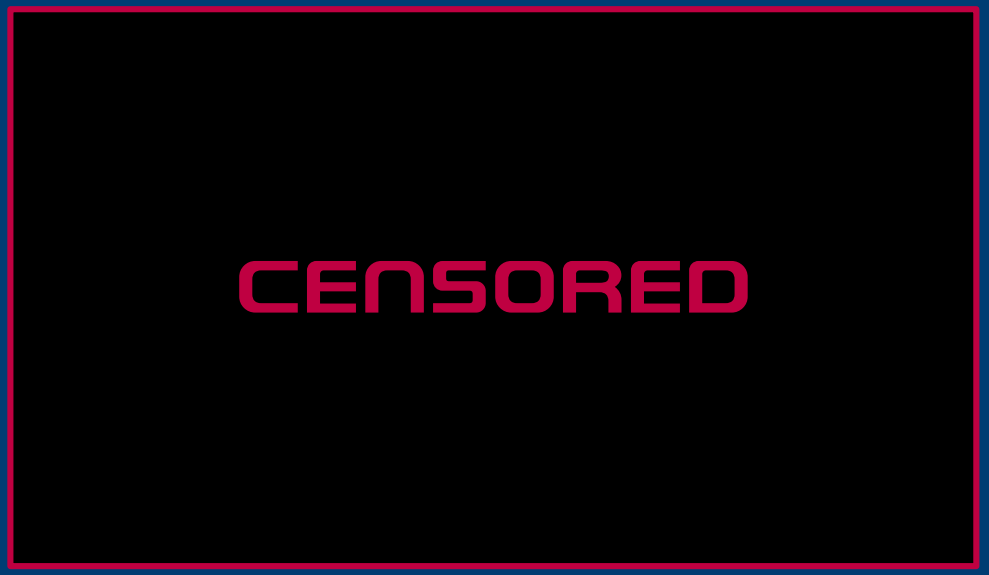
Trigonometric functions + Special functions	`np.sin(…)`
	`np.exp(…)`
	`sp.special.jv(…)`
	`sp.special.zeta(…)`
		`mp.sin(…)`
		`mp.exp(…)`
		`mp.besselj(…)`
		`mp.zeta(…)`
			`mpfr_sin(…)`
			`mpfr_exp(…)`
			`mpfr_jn(…)`
			`mpfr_zeta(…)`

Numerical calculus	Root finding, sum, quadrature (integrals), differentiation, ODE (RK4), Taylor, Fourier, …		Check other libraries: GSL mpack = blas+lapack Boost C++ [www.boost.org]
Linear algebra	Matrix/vector operations (products, inverse, determinant), SVD, linear systems, eigenproblems, matrix functions (exp, cos, …), …		

Exercise: factorial

Let us go back to our factorial implemented on a for loop:

```
def myfactorial(n):
    f = 1
    for i in range(1, n+1):
        f *= i
    return f
```



1) Try to run it for n=30

→ expected result: 265252859812191058636308480000000

2) Now, change your implementation to have the initial f as a float:

→ `f = 1.0`

3) And check again for n=10 and n=30. Float → loss of precision!

4) Change the code to initialize f as a multiple precision float